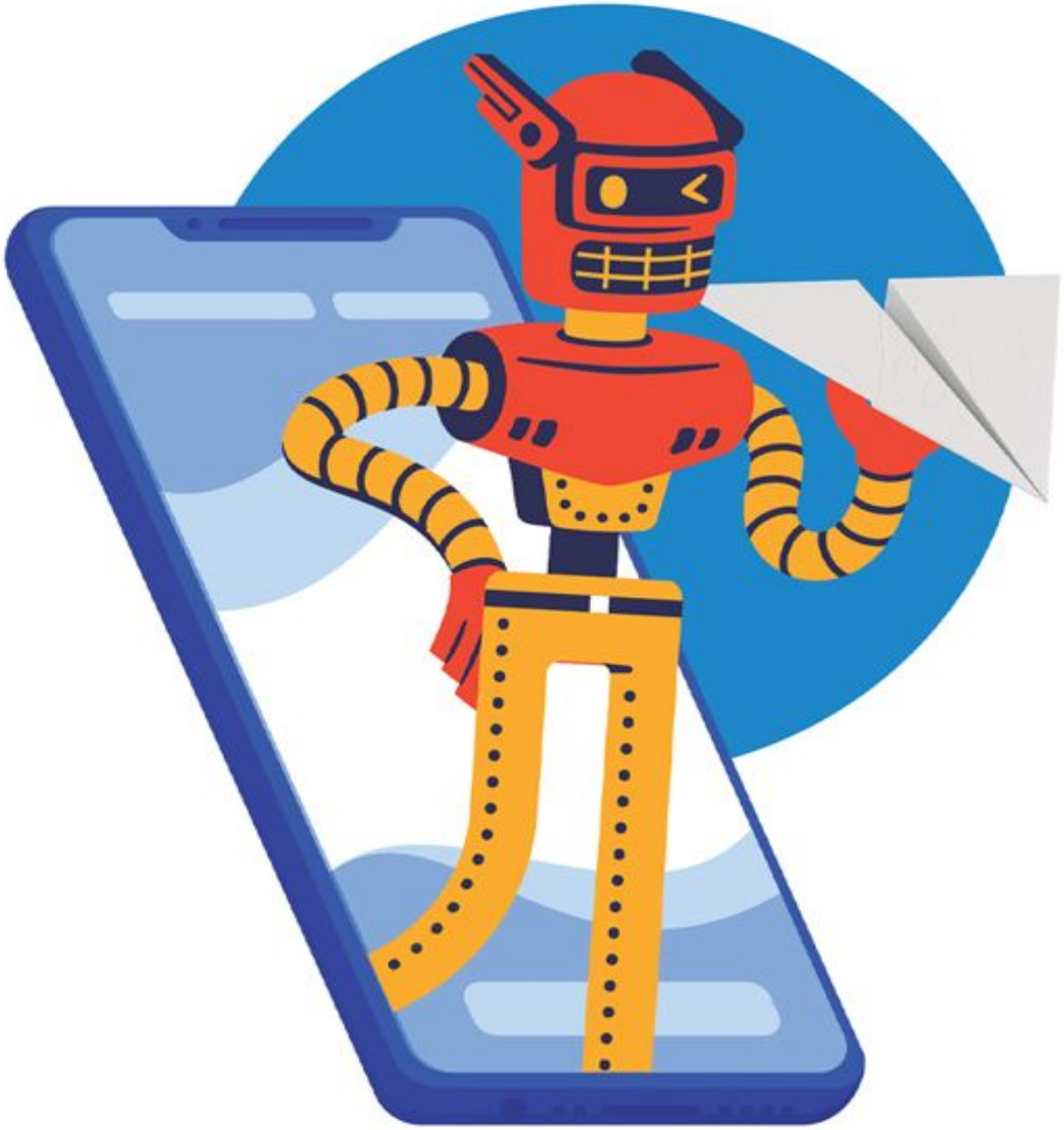
Chatbots para Telegram

Programe seus primeiros bots usando Python



Casa do Código | alura

PRISCILA KELI L. P. FRIZZARIN
FERNANDO BRYAN FRIZZARIN

Sumário

- ISBN
- Agradecimentos
- Sobre os autores
- Introdução
- 1. O que é necessário para continuar
- 2. O Telegram e os chatbots
- 3. O primeiro bot – Olá, Mundo!
- 4. Lendo dados da internet - um bot para previsão do tempo
- 5. Reconhecendo o tipo do arquivo pelo conteúdo
- 6. Imagens e reconhecimento de caracteres
- 7. Recebendo localização – calcular distância entre duas localidades
- 8. Jogo da velha - Usando menus para interação com usuário
- 9. Biscoito da sorte – usando banco de dados
- 10. Conclusão

ISBN

Impresso: 978-85-5519-329-3

Digital: 978-85-5519-328-6

Caso você deseje submeter alguma errata ou sugestão, acesse http://erratas.casadocodigo.com.br.

Agradecimentos

Primeiramente, a Deus, que nos permitiu chegar até aqui, somos muito gratos.

Aos nossos pais, por acreditarem em nós e estarem sempre presentes em nossas vidas.

A nossas famílias, fontes de inspirações, ajuda e amor infinito.

E às pessoas que, com suas ideias e sugestões diferenciadas, nos inspiraram a escrever este livro.

Sobre os autores

Priscila Keli de Lima Pinto Frizzarin

Natural de Araçatuba/SP, é bacharel em Ciência da Computação e pedagoga, especialista em Tecnologia da Informação e Psicopedagogia. Cursou várias disciplinas de mestrado voltadas às áreas de programação e educação. É professora do curso de Desenvolvimento de Sistema do COTIL (Colégio Técnico de Limeira) e, atualmente, é chefe do Departamento de Infraestrutura e Tecnologia dessa mesma instituição de ensino.

As suas áreas de maior interesse são o desenvolvimento de sistemas, tanto para plataforma desktop quanto para dispositivos embarcados, e a inovação. É apaixonada pela docência e busca sempre o aperfeiçoamento.

Fernando Bryan Frizzarin

Casado com a mulher maravilhosa que se apresenta aí acima, é natural de Americana/SP, técnico em Informática, bacharel em Ciência da Computação, psicopedagogo, especialista em Redes de Computadores e MBA em Gestão Estratégica de Negócios.

É gerente de certificações e do suporte técnico da BluePex Cybersecurity e professor do ensino superior na Faculdade de Tecnologia do Estado de São Paulo (FATEC). É, também, voluntário como diretor do Centro de Memória da Inovação da Fábrica de Inovação em Limeira/SP.

Autor dos livros *Arduino: guia para colocar suas ideias em prática*, *Arduino Prático: 10 projetos para executar, aprender, modificar e*

dominar o mundo e *NodeMCU: 15 passos para se tornar um mestre em IoT* — todos publicados pela Editora Casa do Código.

Autor de vários softwares, desenhos industriais e marcas registradas no INPI, é coautor da patente BR1020140270159/2014: *Dispositivo automatizado de liberação controlada*, projeto desenvolvido em conjunto com os alunos Bianca de Mori Machado Amaral e Felipe Ferreira da Silva, incluindo apoio da arquiteta Marylis Barbosa de Souza. Esse projeto foi desenvolvido nas disciplinas de Desenvolvimento de Projetos e Tópicos Avançados em Informática no Colégio Técnico de Limeira (Unicamp). O depósito foi feito por meio da Agência de Inovação da Unicamp, a Inova.

Sua grande área de interesse é a Administração, a inovação e todas as demais áreas adjacentes, desde capital humano até programação para dispositivos embarcados, além de toda a magnitude de disciplinas, matérias e conhecimentos que isso implica. Adora disseminar conhecimento em palestras e aulas — e, também, em um bom papo descompromissado. Basta um convite!

Introdução

Atualmente, ao entrar em contato com alguma empresa ou instituição, é comum nos depararmos com chats nos quais não conversamos com outra pessoa, mas sim com robôs. São os *chatbots*: uma ferramenta de respostas rápidas que auxilia as empresas no relacionamento com seus clientes.

Esses chatbots podem ser programados para o serviço de atendimento ao consumidor (SAC), para a captação de novos clientes ou para atuarem como otimizadores. No atendimento ao cliente, os chatbots são programados para responder dúvidas frequentes, receber reclamações, entre outras funções. Já os chatbots para captação de novos clientes simulam o atendimento humano na apresentação dos produtos ou serviços oferecidos pela empresa quando o usuário está visitando o site. E os chatbots otimizadores são muito utilizados em e-commerce e apps para ajudar o cliente na realização de tarefas como comprar e tirar dúvidas. Além disso, podemos ter chatbots conversacionais, ou seja, chatbots para interagir com os usuários como forma de entretenimento, entre outros usos.

Já o Telegram é um aplicativo gratuito para bate-papo por meio do qual os usuários podem enviar mensagens de texto, vídeos, fotos etc. Criado na Rússia em 2013, este aplicativo ganhou visibilidade em 2015 no Brasil. Algumas de suas vantagens são: a não disponibilização do número de telefone do usuário, a possibilidade de criação de grupos com até 200 mil membros e a inexistência de limites para o tamanho dos chats e das mídias enviadas.

Em janeiro de 2021, o Telegram ultrapassou a marca de 500 milhões de usuários ativos, tornando-se uma ferramenta muito popular e bastante usada no meio empresarial na criação de enquetes, no atendimento ao cliente, além de permitir a criação de bots com

finalidades diversas. No meio educacional, após o surgimento da pandemia de COVID-19, essa ferramenta vem sendo utilizada para agilizar a comunicação do professor com alunos e pais, tornando-se uma facilitadora da comunicação escolar.

A proposta deste livro é criar chatbots utilizando a plataforma Telegram.

Não há necessidade de ter um profundo conhecimento da linguagem de programação Python. Espera-se apenas que você saiba lógica de programação e como programar o suficiente para instalar e configurar um aplicativo para Windows e Linux. Também serão necessários conhecimentos básicos em linguagem SQL, pois usaremos comandos para manipular bancos de dados.

Este livro tratará de como fazer essa tarefa nas plataformas Windows e Linux, que são as mais populares, mas você não terá problemas em adaptá-las para o ambiente Apple, por exemplo. Todos os códigos-fontes que serão apresentados são os mesmos para qualquer plataforma, desde que tenham o Python devidamente instalado e em funcionamento.

Capítulo 1

O que é necessário para continuar

1.1 A linguagem Python

A linguagem Python é *open source*, ou seja, seu código-fonte está disponível para que qualquer usuário possa modificá-lo para diversos fins. Portanto, não existe apenas uma implementação desta linguagem. Quem mantém uma implementação oficial da linguagem é a Python Software Foundation, uma organização sem fins lucrativos.

A implementação padrão e mais conhecida do Python está escrita em linguagem C, conhecida como CPython, mas há outras, como IronPython, .NET, Jython e PyPy.

Esta é uma linguagem interpretada, o que quer dizer que o código-fonte é compilado e, a partir disso, um arquivo em linguagem de máquina é gerado. Esse arquivo será interpretado por uma máquina virtual responsável pela execução, conhecida como PVM (*Python Virtual Machine*, em português, máquina virtual Python).

A vantagem desse processo é que o arquivo gerado pode ser lido em qualquer arquitetura, o que faz com que esta linguagem seja multiplataforma. Mas o que isso significa?

Quando falamos em arquitetura, estamos nos referindo aos processadores, que podem possuir arquitetura de 32 bits ou de 64 bits. Processadores de 32 bits conseguem processar sequências de dados de até 32 bits (bit é a unidade de medida que o computador usa: bit, byte, Kilobyte, Megabyte, Gigabyte, Terabyte etc.). Já os de

64 bits conseguem processar o dobro de instruções em relação ao de 32 bits. Além disso, processadores de 32 bits conseguem gerenciar, no máximo, 3,2 GB de memória RAM; já os processadores de 64 bits vão muito além.

Dizer que a linguagem é multiplataforma significa que o arquivo gerado em linguagem de máquina será reconhecido em qualquer sistema operacional, por exemplo, Windows ou Linux.

Características do Python

Algumas características do Python que são importantes de conhecermos brevemente neste ponto de partida são: tipagem dinâmica, a ausência de delimitadores de bloco de comando e a Orientação a Objetos.

Na tipagem dinâmica, as variáveis são criadas dinamicamente, não havendo necessidade de declaração prévia. O interpretador da linguagem Python analisa, de acordo com o valor atribuído à variável, qual o tipo que ela assumirá.

Com relação aos delimitadores de bloco, em vez de utilizá-los como na linguagem C, que usa chaves para definir blocos de comando, a linguagem Python organiza seu código através da *indentação*, que nada mais é que uma forma de organizar o código a fim de auxiliar na estruturação e leitura. Para isso, são acrescentados espaços no início das linhas de código para auxiliar a manter uma hierarquia visual.

No caso de uma estrutura `if`, o que determina se uma instrução está condicionada ao fato de a estrutura `if` ser verdadeira é a indentação. Vejamos o exemplo:

```
if mensagem == 'Oi':
    bot.sendMessage(chat_id,"Olá, Mundo!")
```

Nesse caso, a instrução `bot.sendMessage(chat_id,"Olá, Mundo!")` será executada se a condição `if` for verdadeira.

Agora, observe este outro exemplo:

```
if mensagem == 'Oi':
bot.sendMessage(chat_id,"Olá, Mundo!")
```

Nesse caso, a instrução `bot.sendMessage(chat_id,"Olá, Mundo!")` será executada independente de a condição verificada no `if` ser verdadeira.

A **Orientação a Objetos** é o que chamamos de paradigma, ou seja, um padrão que a linguagem de programação Python utiliza. Ele define que a linguagem de programação Python trabalhe com conceitos de classes e objetos. A *classe* define a estrutura de um determinado objeto sem especificar valores, podendo ser qualquer coisa. Esse objeto é composto por *atributos* e *métodos*: atributos são informações referentes a ele e métodos são comportamentos que esse objeto pode ter.

Como exemplo, vamos considerar o objeto "robô". Quais atributos, ou seja, quais características podemos definir para esse robô? Podemos pensar, por exemplo, em tamanho e posição. E métodos? Movimentar-se e limpar o chão, por exemplo. Então temos a seguinte definição:

```
"Robô"                          --------> Identificador da classe
Tamanho, posição                --------> Atributos
Movimentar-se, limpar o chão    --------> Métodos
```

A esse conjunto de informações, atributos e métodos, damos o nome de "classe". Portanto, a partir do meu exemplo, tenho uma classe chamada "robô" e posso definir um objeto chamado "Limpador", que possui um tamanho e uma posição e que pode se movimentar e limpar o chão.

Para desenvolver os projetos na linguagem Python, você pode utilizar uma IDE (*Integrated Development Environment* – em português, Ambiente Integrado de Desenvolvimento). Elas são ferramentas que facilitam a programação por apresentarem um ambiente de desenvolvimento mais amigável.

Em uma IDE você codifica e interpreta os programas desenvolvidos. Exemplos de IDEs são o IDLE Shell, o VSCode (Visual Studio Code) e o ATOM.

Ainda não se convenceu de que Python é uma linguagem que vale a pena? Então vou lhe dar mais um motivo: essa linguagem tem ganhado visibilidade ao longo dos anos e, atualmente, é uma das mais utilizadas, junto às linguagens C/C# e Java.

Devido à popularização da linguagem Python e das diversas aplicabilidades que vão surgindo, as versões estão em constantes atualizações, mas isso não afetará os exemplos apresentados.

1.2 Nosso ambiente de trabalho

Para a execução de todos os códigos-fontes apresentados neste livro, serão necessárias algumas instalações. O primeiro passo é definir em qual ambiente você trabalhará: Windows ou Linux.

Escolhido o ambiente, é só seguir o passo a passo descrito a seguir e colocar a mão na massa.

No ambiente Microsoft Windows

Em ambiente Windows, o primeiro passo é instalar o Python. Para isso, devemos acessar a página da Python Software Foundation (disponível em www.python.org).

Na seção de downloads, há a opção de baixar a última versão do Python para Windows. A importância de começar um novo projeto com a última versão da linguagem de programação é a garantia de ter atualização e suporte por mais tempo (mas não para sempre).

Uma vez feito o download do arquivo, basta executá-lo para começar a instalação. No geral, não tem nenhum segredo. É necessário apenas marcar a opção *"Add Python 3.9 to PATH"* na primeira janela da instalação, como na imagem a seguir:

Figura 1.1: Opção "Add Python 3.9 to PATH" deve estar selecionada durante a instalação

Após a instalação ser concluída, é importantíssimo testarmos para ver se está tudo funcionando. Procure no menu Iniciar por `IDLE Python` e execute-o. O IDLE é um ambiente de desenvolvimento integrado para o Python. É nele que escreveremos e executaremos nossos códigos.

Executando o IDLE, você terá uma janela de Shell para o Python. Rapidamente, para verificar se está tudo funcionando, vamos tentar um "Olá, Mundo!". O código-fonte será apenas:

```
print("Olá, Mundo!")
```

Se você tiver como resposta um "Olá, Mundo!" de volta na linha de baixo, conforme a imagem a seguir, está tudo ok.

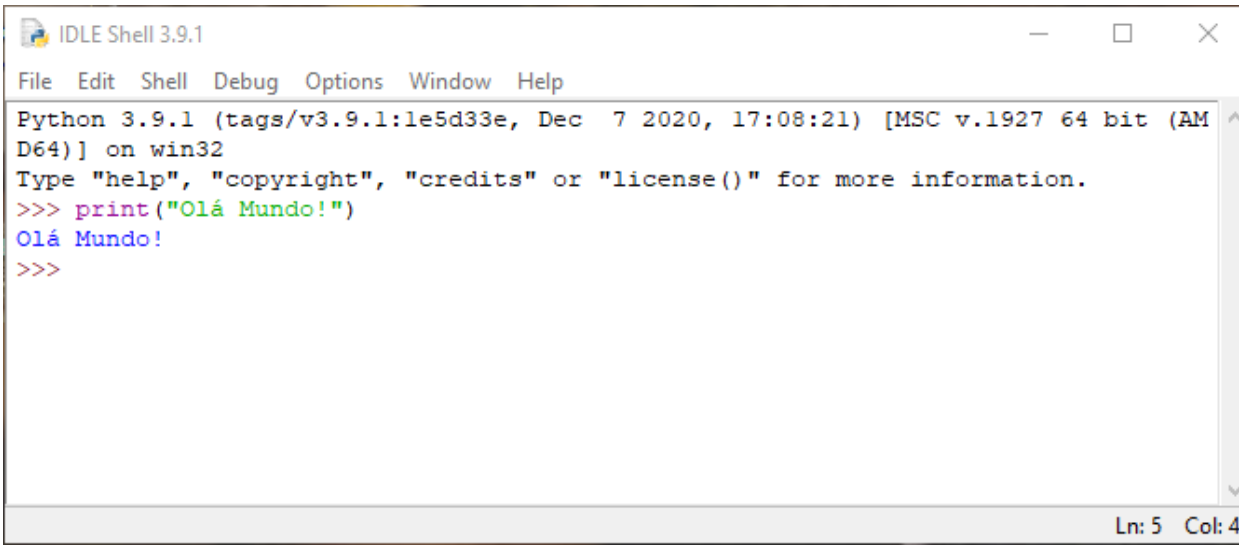


Figura 1.2: "Olá, Mundo!" indicando que a instalação está correta

Agora vamos instalar a biblioteca **Telepot**, que contém um conjunto de objetos, classes e atributos específicos para lidar com o Telegram. Ela funciona, nas versões atuais, com o Python 2.7 e o 3. Vamos abordá-la com mais detalhes mais adiante.

Para instalarmos o Telepot, execute o Prompt de Comando do Windows como administrador. Nele, vamos usar o `pip` (*Python Package Index* – PyPI), que é um gerenciador de pacotes para o Python, para instalar a biblioteca Telepot. A linha de comando para isso é:

```
pip install telepot
```

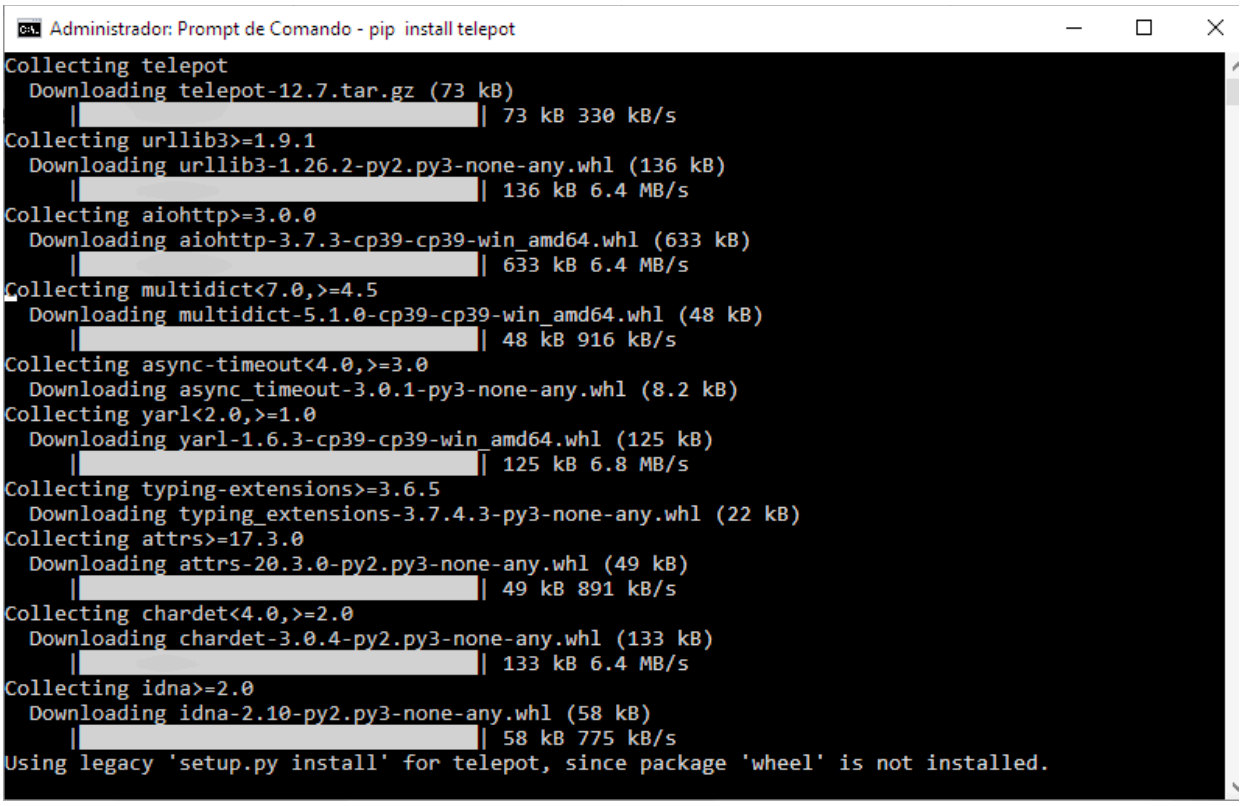


Figura 1.3: Instalação da biblioteca Telepot em andamento

Se você estiver em uma rede com proxy, você terá que digitar duas linhas de comando antes:

```
set
http_proxy=http://usuário:senha@endereço_do_proxy:porta_do_proxy
set
https_proxy=https://usuário:senha@endereço_do_proxy:porta_do_pro
xy
```

Onde se lê `usuário`, substitua pelo nome do seu usuário no proxy. No lugar de `senha`, digite a sua senha para acessar o proxy. Troque `endereço_do_proxy` pelo endereço IP do proxy da sua rede e também troque `porta_do_proxy` pela porta do proxy da sua rede.

Depois de configurar o proxy, é só instalar o Telepot com o comando já apresentado:

```
pip install telepot
```

Para testar se está tudo instalado e funcionando como deve ser, volte ao IDLE Python e digite:

```
import telepot
```

Se após dar `ENTER` o Python não retornar nenhuma mensagem de erro, é sinal de que tudo ocorreu com sucesso. Caso contrário, volte ao primeiro passo e verifique se todos foram cumpridos corretamente.

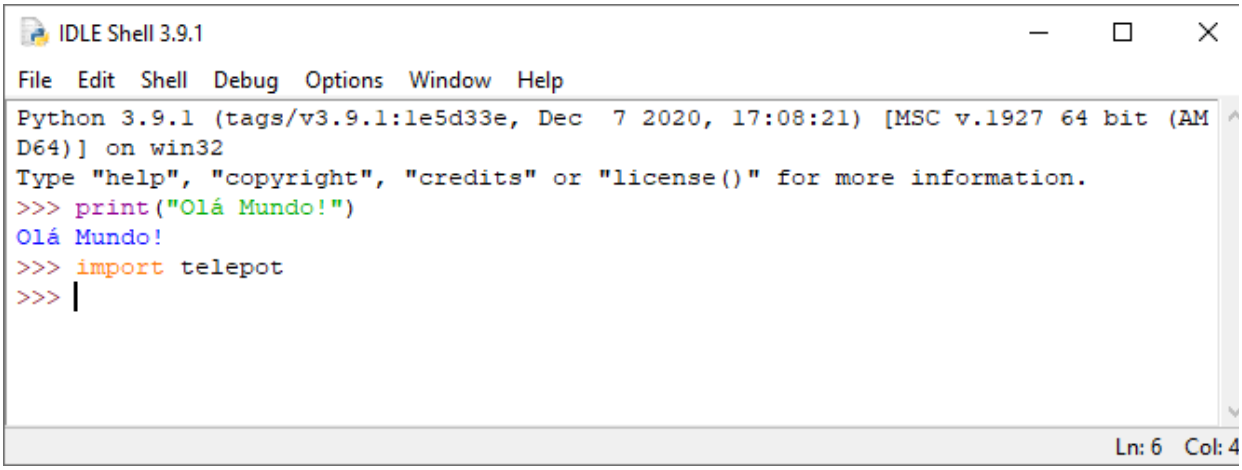


Figura 1.4: A biblioteca Telepot funcionando no Python corretamente

No ambiente Linux

Há diversas distribuições e versões de Linux disponíveis e cada pessoa pode (e deve) usar a de sua preferência, mas, para instalar o Python e a biblioteca Telepot, vamos considerar as distribuições do tipo Debian, tais como as utilizadas no Raspberry Pi. Essas distribuições usam o `apt-get` como gerenciador de pacotes.

Nas distribuições Linux recentes, o Python já vem pré-instalado por padrão, mas é sempre bom se prevenir caso você escolha uma distribuição ou uma versão de alguma distribuição em que isso não seja uma verdade.

Também vamos considerar que você está usando Linux com o console do sistema aberto ou mesmo sem interface gráfica. Vale lembrar que, caso esteja usando a interface gráfica, você pode abrir um console de comandos para executar os comandos a seguir e, para a utilização, você pode escolher se continua no console ou usa a interface gráfica, que tem o mesmo funcionamento e aspecto do Windows.

Para instalar o Python no ambiente Linux, usamos o seguinte comando:

```
apt-get install python3
```

Em alguns sistemas Linux, é necessário ter permissões de superusuário (*root*) para realizar a instalação de pacotes via `apt-get`. Se este for o seu caso, adicione `sudo` antes do `apt-get`:

```
sudo apt-get install python3
```

Caso sua rede utilize proxy, será necessário configurá-lo para que o `apt-get` funcione. Para isso, são necessários alguns pequenos passos.

Primeiro, crie um arquivo chamado `10proxy` no diretório `/etc/apt/apt.conf.d/`. Você pode fazer isso com o comando:

```
nano /etc/apt/apt.conf.d/10proxy
```

Lembre-se de que em alguns sistemas Linux é necessário ter permissões de superusuário (*root*) para realizar a criação deste arquivo. Se este for o seu caso, adicione `sudo` antes do `nano`:

```
sudo nano /etc/apt/apt.conf.d/10proxy
```

Neste arquivo deverá constar a configuração de acesso via proxy, que é apenas uma linha, com o seguinte formato:

```
Acquire::http::Proxy
"http://usuário:senha@endereço_do_proxy:porta_do_proxy";
```

Da mesma forma que no modelo para ambiente Windows, onde se lê `usuário` e `senha`, coloque o seu nome de usuário para autenticação no proxy e a sua senha, respectivamente. Troque o campo `endereço_do_proxy` pelo endereço IP do proxy de sua rede e troque `porta_do_proxy` pelo número da porta que seu proxy usa.

Realizadas essas trocas, a linha ficará, por exemplo, com o seguinte aspecto:

```
Acquire::http::Proxy "http://fernando:123abc@192.168.0.1:1234";
```

Lembre-se sempre do **ponto e vírgula** no final, pois ele indica o fim do comando.

Podemos testar o funcionamento do Python no Linux usando o comando `python3` e pressionando `ENTER`:

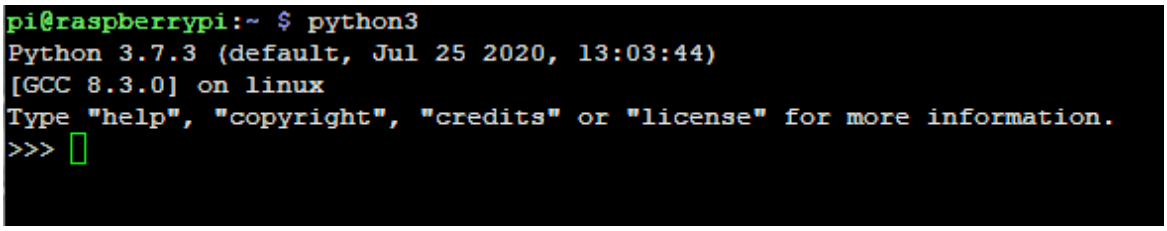


Figura 1.5: Executando Python pelo console de um sistema Linux

Da mesma forma que fizemos no ambiente Windows, podemos executar um comando `print` para testar com um "Olá, Mundo!":

```
print("Olá, Mundo!")
```

Caso retorne para você um `Olá, Mundo!` como na imagem a seguir, e não uma mensagem de erro, quer dizer que o Python está em perfeito e pleno funcionamento.

```
pi@raspberrypi:~ $ python3
Python 3.7.3 (default, Jul 25 2020, 13:03:44)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Hello World!")
Hello World!
>>>
```

Figura 1.6: Resultado do comando print

Para sair da linha de comandos do Python, use o comando `quit()`.

Tudo certo com nosso Python! Vamos agora instalar a biblioteca Telepot. Antes, temos que instalar o utilitário `pip`. Fazemos isso com o comando:

```
apt-get install python3-pip
```

```
pi@raspberrypi:~ $ sudo apt-get install python3-pip
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  dh-python gir1.2-glib-2.0 libexpat1-dev libgirepository-1.0-1 libpython3-dev
  libpython3.7-dev python-pip-whl python3-asn1crypto python3-cffi-backend
  python3-crypto python3-cryptography python3-dbus python3-dev
  python3-distutils python3-entrypoints python3-gi python3-keyring
  python3-keyrings.alt python3-lib2to3 python3-secretstorage
  python3-setuptools python3-wheel python3-xdg python3.7-dev
Suggested packages:
  python-crypto-doc python-cryptography-doc python3-cryptography-vectors
  python-dbus-doc python3-dbus-dbg gnome-keyring libkf5wallet-bin
  gir1.2-gnomekeyring-1.0 python-secretstorage-doc python-setuptools-doc
The following NEW packages will be installed:
  dh-python gir1.2-glib-2.0 libexpat1-dev libgirepository-1.0-1 libpython3-dev
  libpython3.7-dev python-pip-whl python3-asn1crypto python3-cffi-backend
  python3-crypto python3-cryptography python3-dbus python3-dev
  python3-distutils python3-entrypoints python3-gi python3-keyring
  python3-keyrings.alt python3-lib2to3 python3-pip python3-secretstorage
  python3-setuptools python3-wheel python3-xdg python3.7-dev
0 upgraded, 25 newly installed, 0 to remove and 0 not upgraded.
Need to get 51.4 MB of archives.
After this operation, 84.3 MB of additional disk space will be used.
Do you want to continue? [Y/n]
```

Figura 1.7: Instalando o pip

Na sequência, usamos o seguinte comando para instalar definitivamente o Telepot:

```
pip3 install telepot
```

Note que usamos `pip3` para fazer a instalação. Esse comando identifica, no ambiente Linux, a última versão a ser utilizada. Já no ambiente Windows, usamos simplesmente `pip` e ele já entende que é para baixar a última versão.

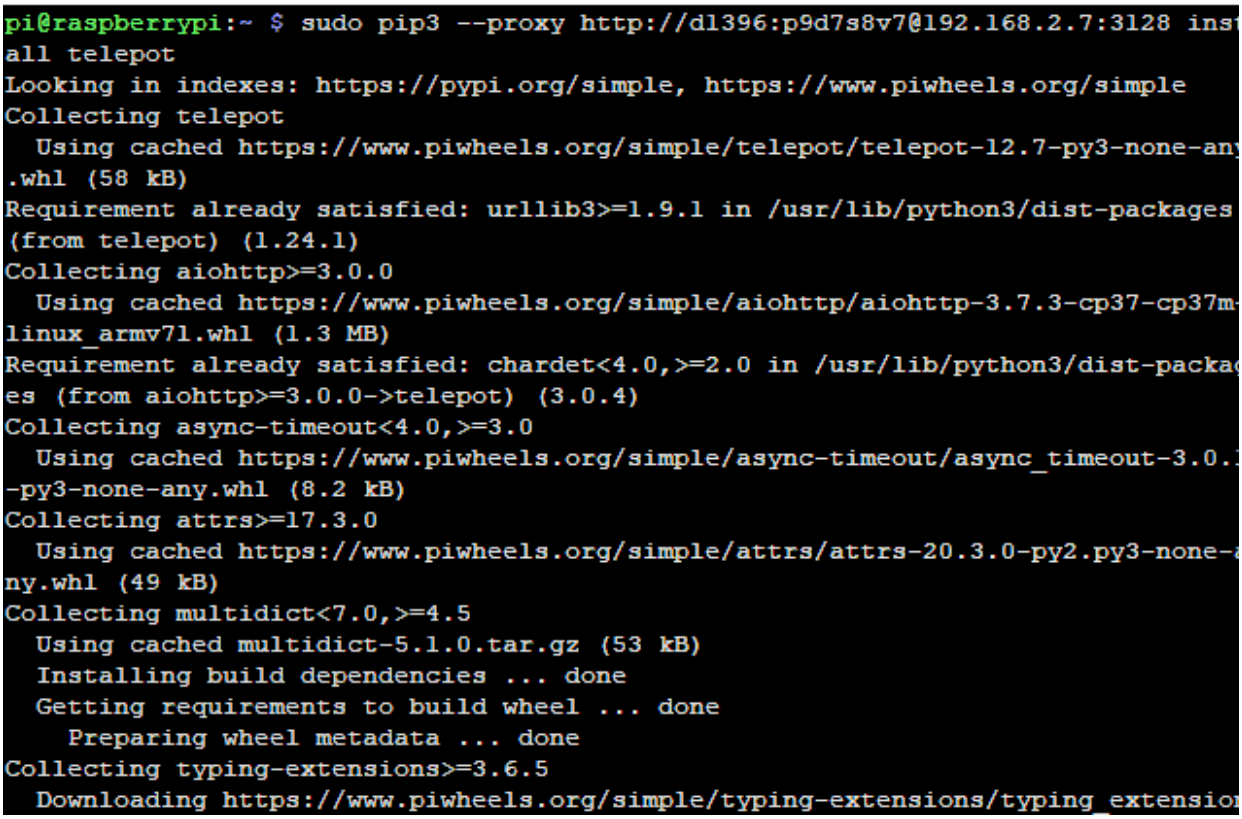


Figura 1.8: Instalando a biblioteca Telepot

Caso você esteja em uma rede que tem um proxy, será necessário adicionar as informações dele na linha de comandos. Dessa forma, elas deverão seguir os modelos apresentados a seguir.

Para exportar as variáveis de ambiente para o proxy:

```
export
http_proxy="http://usuário:senha@endereço_do_proxy:porta_do_prox
y"
export
https_proxy="http://usuário:senha@endereço_do_proxy:porta_do_pro
xy"
```

Para instalar o Telepot via `pip3` usando proxy:

```
pip3 -proxy
http://usuário:senha@endereço_do_proxy:porta_do_proxy install
telepot
```

Assim como fizemos anteriormente, substitua os respectivos campos pelos dados corretamente. Lembre-se que, dependendo do seu sistema operacional, talvez você tenha que ter permissões de superusuário e usar o comando `sudo`.

Uma vez que esteja tudo instalado, vamos fazer um teste para saber se está tudo funcionando. Execute novamente o Python:

```
python3
```

No prompt de comandos do Python, execute o seguinte comando:

```
import telepot
```

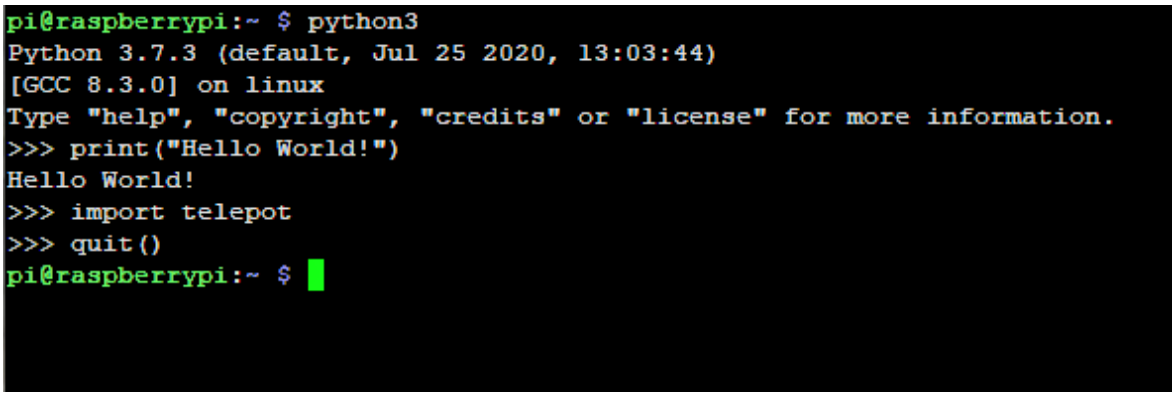


Figura 1.9: Teste de funcionamento

Se ao fazer isso você não receber nenhuma mensagem de erro (aliás, ele não deve retornar nenhuma mensagem, como mostra a imagem

anterior), quer dizer que está tudo funcionando normalmente.

Lembre-se de que o comando para sair do prompt de comandos do Python é `quit()`.

Banco de dados

Por fim, será necessária a instalação de um banco de dados. Seja qual for o sistema operacional que você estiver utilizando, aqui usaremos o banco de dados MariaDB, que é uma ferramenta de gerenciamento de banco de dados compatível tanto com o Windows quanto com o Linux.

O primeiro passo é acessar o site da MariaDB Foundation, disponível em www.mariadb.org. Em seguida, clique no link de `Download` e selecione a versão mais atual, o sistema operacional, a arquitetura e o tipo de pacote.

Agora que você já tem as informações básicas sobre a linguagem de programação que será utilizada nos exemplos deste livro, vamos conhecer um pouco sobre o Telegram e os chatbots?

Capítulo 2

O Telegram e os chatbots

Atualmente, quando se deseja esclarecer uma dúvida, fazer uma sugestão ou mesmo uma reclamação sobre algum serviço ou produto, muitas vezes você tem a opção de usar o chat para isso. Vamos combinar que ficar horas e horas no telefone à espera de atendimento – e por vezes não conseguir resolver o problema – não é nada agradável. Para ter maior eficiência e velocidade no atendimento, as empresas têm optado pelo uso de chats automatizados — os chatbots.

Chatbot é um programa de computador que interage com os seus usuários através de mensagens de texto automatizadas. É uma ferramenta de bate-papo.

Os chatbots podem ser baseados em regras (simples) ou em inteligência artificial (complexos). A proposta deste livro é ensinar a você como construir chatbots simples, ou seja, chatbots baseados em regras.

O funcionamento é simples. Na escrita do código haverá palavras-chaves predefinidas pela pessoa programadora e, conforme o usuário escreve uma mensagem, o programa saberá identificar essas palavras-chaves e responder.

E o que as pessoas ganham com o uso de chatbots? Clientes de uma empresa, por exemplo, conseguem respostas mais rápidas às suas indagações (não ouvirão, por exemplo, "todos os nossos atendentes estão ocupados"). As empresas, por sua vez, ganham em prestar um atendimento ágil, personalizado e sem interrupções — fatores que diminuem a probabilidade de que um cliente, mesmo insatisfeito, vá trocar uma empresa pela concorrente.

Aliada a essa automatização, há uma plataforma que, entre tantas vantagens, não requer um número de telefone para configurar — o Telegram.

O **Telegram** é uma ferramenta criada em 2013 para troca de mensagens de forma rápida e segura. Todas as informações ficam armazenadas na nuvem, ou seja, nos servidores da empresa.

Existem várias vantagens no uso dessa ferramenta, além da rapidez e segurança. Entre elas, podemos destacar que: é uma ferramenta simples, de interface intuitiva; possui uma Bot API (*Application Programming Interface*, em português, Interface de Programação de Aplicações) aberta, o que permite a criação de bots no Telegram; e que ela não restringe o tamanho das mídias enviadas, bem como o tamanho dos chats.

Para utilizar a Bot API, é necessário criar uma conta, que não precisa de um número de telefone nem de autorização prévia para ser configurada, mas é necessário também estar atento aos Termos de Uso do Telegram (disponível em https://telegram.org/tos).

A interação com o bot pode ocorrer de duas formas diferentes: pelo envio de mensagens a partir de uma conversa chat com o bot ou pelo envio de requisições ao bot através do campo de texto, usando o comando `@nomedoseubot` e a instrução que você quer que seja executada por ele.

Os exemplos deste livro trazem ideias de interação dos usuários do Telegram com chatbots para inspirá-los.

Vamos começar?

Criando um chatbot no Telegram

Há praticamente duas maneiras de criar um chatbot: usando o app do Telegram em seu smartphone ou usando o Telegram Web, a

versão para desktops.

Em todas as tarefas deste livro, vamos utilizar um dispositivo móvel (smartphone), mas os comandos e caminhos são os mesmos para qualquer interface para o Telegram, seja em iOS, em MacOS, no Android, na Web, no Windows ou mesmo em Linux.

Sendo assim, para começarmos, instale o app em seu smartphone e o execute. A tela inicial do aplicativo, ainda sem nenhum chat ou chatbot ativo, é esta:

Editar

Chats

Mensagens ou usuários

Você ainda
não tem conversas.

Nova Mensagem

Contatos

Chamadas

Chats

Configurações

Figura 2.1: O aplicativo do Telegram aberto no celular, mas ainda sem qualquer mensagem.

Na barra de pesquisa, digite e busque pelo perfil `@BotFather`. O @BotFather é o chatbot do Telegram para lidar com a criação, a configuração e, se necessário, a exclusão de seus chatbots.

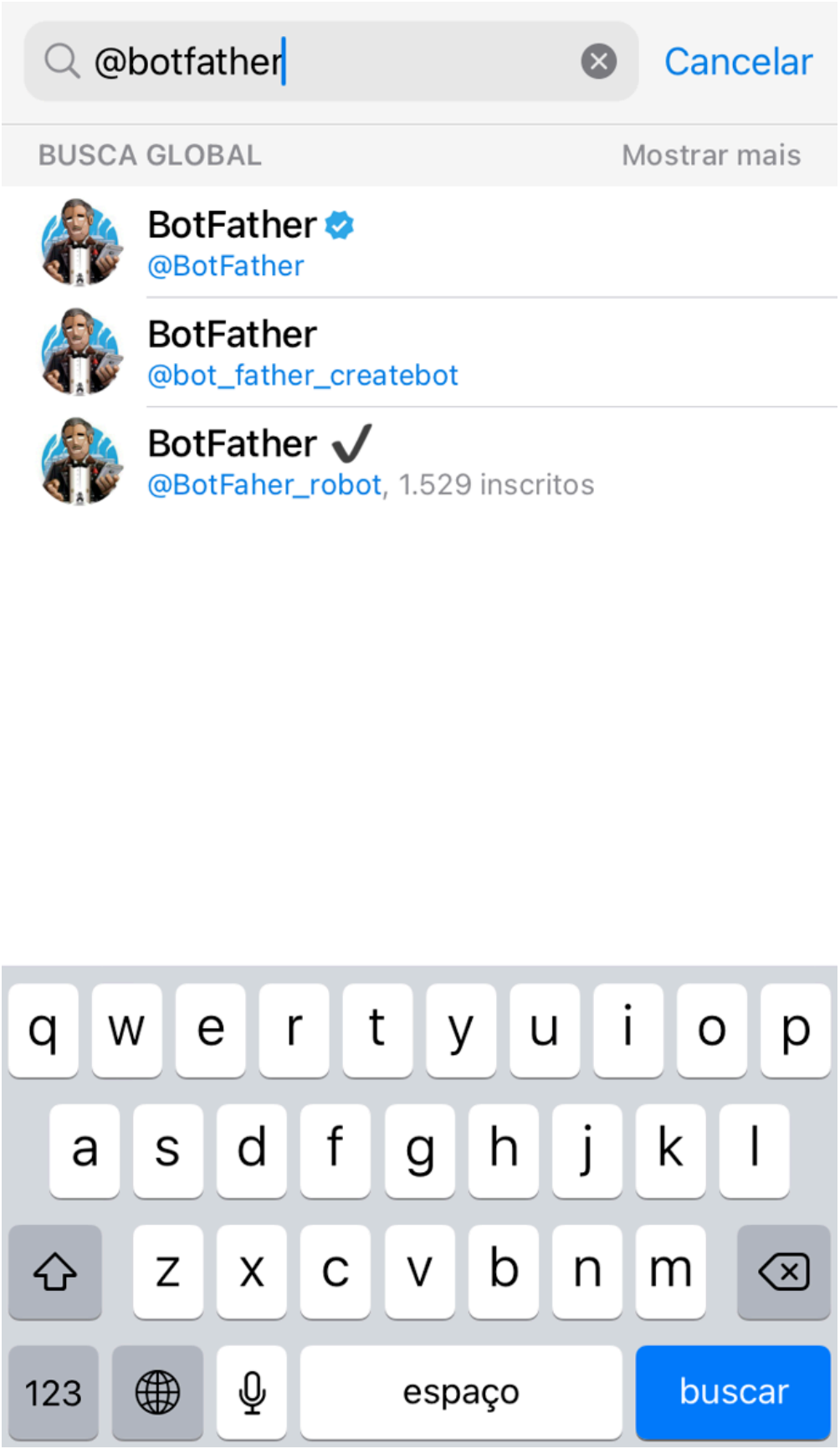

@botfather
Cancelar
BUSCA GLOBAL
Mostrar mais
BotFather
@BotFather
BotFather
@bot_father_createbot
BotFather
@BotFaher_robot, 1.529 inscritos
q w e r t y u i o p
a s d f g h j k l
z x c v b n m
123
espaço
buscar

Figura 2.2: A busca pelo @BotFather via barra de pesquisa no app do Telegram

Tenha certeza de que encontrou o usuário correto. O @BotFather é um usuário certificado, ele possui o ícone de verificação, que é um símbolo azul ao lado direito do nome, conforme mostra a imagem a seguir:

Figura 2.3: Usuário @BotFather com o ícone de usuário verificado

Clique sobre ele e você verá uma mensagem com o título "O que este bot pode fazer?". No rodapé da tela, você terá o botão "Começar". É só tocar/clicar nele para começar a criação de um chatbot.

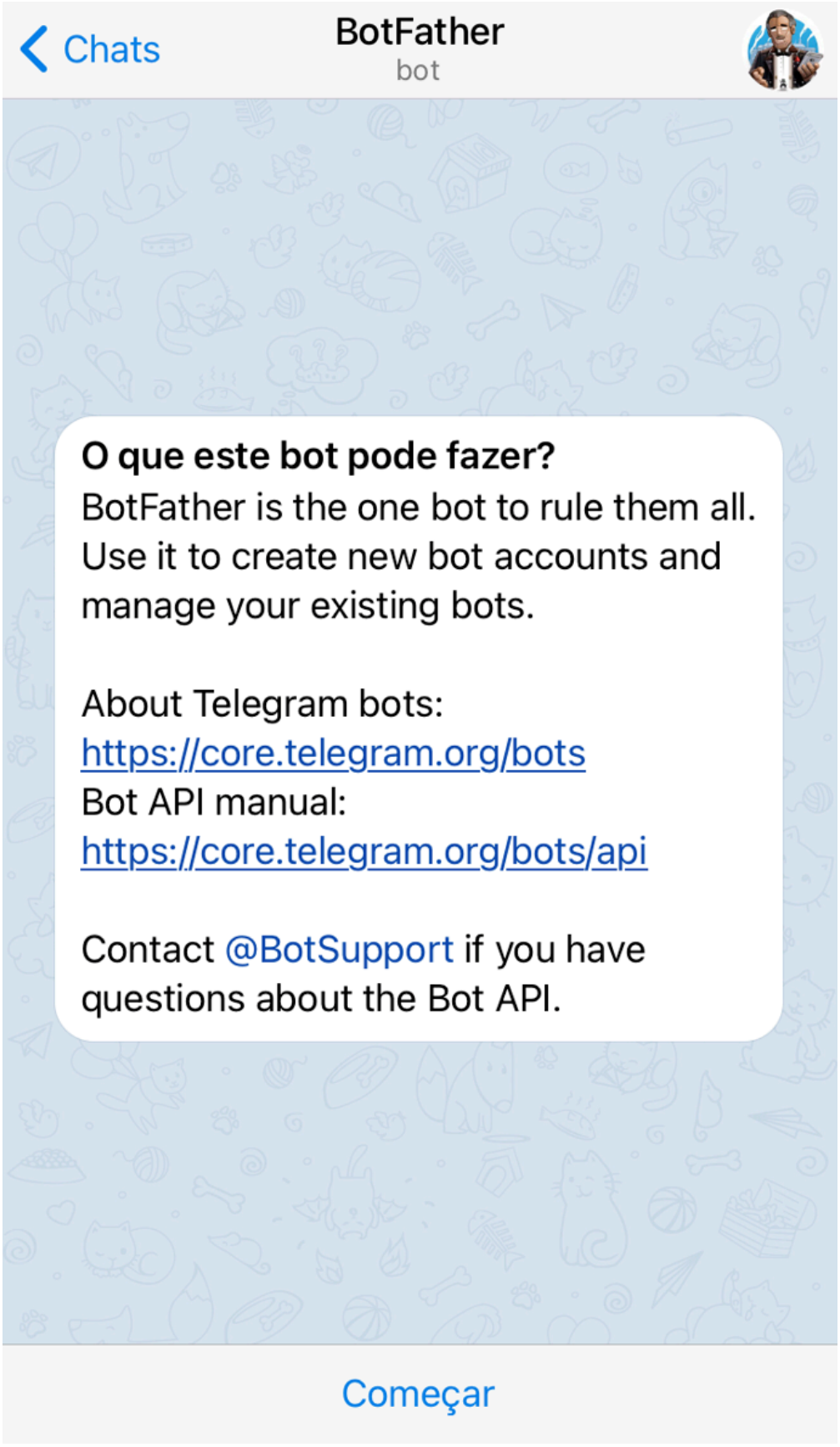

Chats
BotFather
bot
O que este bot pode fazer?
BotFather is the one bot to rule them all. Use it to create new bot accounts and manage your existing bots.
About Telegram bots:
https://core.telegram.org/bots
Bot API manual:
https://core.telegram.org/bots/api
Contact @BotSupport if you have questions about the Bot API.
Começar

Figura 2.4: Tela inicial do @BotFather com o botão “Começar” no rodapé

Após tocar/clicar em "Começar", o @BotFather mostrará todos os comandos possíveis. Vamos a uma breve explicação sobre o que cada um desses comandos faz.

Para criar e editar bots:

- `/newbot` - Criar um novo bot
- `/mybots` - Editar as configurações dos seus bots. Ainda em versão beta.
- `/setname` - Mudar o nome do bot.
- `/setdescription` - Mudar a descrição do bot.
- `/setabouttext` - Incluir informações sobre seu bot, como detalhes de funcionamento ou regras de uso.
- `/setuserpic` - Incluir ou alterar a imagem do perfil do bot.
- `/setcommands` - Incluir informação sobre os comandos que seu bot responderá, se houver — por exemplo, para executar alguma ação ou configuração para o usuário. Mesmo que você programe algum comando específico para seu bot mostrar uma ajuda, é importante que você coloque esse comando nessa lista, para facilitar a vida dos seus usuários.
- `/deletebot` - Apagar um bot.

Para configurar bots:

- `/token` - Gerar um token de autorização. O token é importantíssimo para o funcionamento do bot. O seu bot será identificado através desse token.
- `/revoke` - Remover o token de autorização do bot. É sempre importante usar esse comando antes de apagar o bot, para que, além de eliminar o bot, também seja removido o token.

- `/setinline` - Habilitar ou desabilitar a capacidade do bot de receber comandos a partir de qualquer conversa. Por exemplo, quando você está em uma conversa com alguma pessoa e digita um comando *inline* do seu bot, mesmo que ele não participe da conversa.
- `/setinlinegeo` - Habilitar ou desabilitar requisições *inline* de georreferência para seu bot. Isso fará com que o bot solicite aos usuários a localização geográfica deles a cada vez que um comando for enviado para ele. Essa é uma opção interessante, já que seu bot pode dar determinada resposta baseando-se no contexto geográfico.
- `/setinlinefeedback` - Habilitar ou desabilitar as opções de receber resultados fornecidos pelos usuários do seu bot enviados por meio de seus bate-papos.
- `/setjoingroups` - Habilitar ou desabilitar a opção de o bot poder ser incluído em grupos de bate-papo. Com isso, você pode permitir que o usuário converse diretamente com seu bot ou que este seja submetido a uma conversa "comunitária" em um grupo.
- `/setprivacy` - Habilitar ou desabilitar o modo de privacidade para grupos. Quando ativo, o bot apenas responderá a comandos iniciados com a barra `/` ou pelo seu nome. Quando inativo, o bot poderá responder a qualquer mensagem enviada pelos usuários, sem a necessidade de iniciar a mensagem com a barra `/`. Caso esteja construindo um bot para participar de conversas, é pertinente que essa configuração esteja desabilitada, assim ele interagirá com uma necessidade menor de protocolos a serem seguidos.

Para jogos:

- `/mygames` - Editar as configurações de um jogo que tenha criado. Os bots no Telegram podem oferecer aos usuários jogos feitos em HTML5 e JavaScript.
- `/newgame` - Criar um novo jogo.

- `/listgames` - Listar os jogos já criados por você.
- `/editgame` - Editar as configurações de um jogo.
- `/deletegame` - Apagar um jogo já existente.

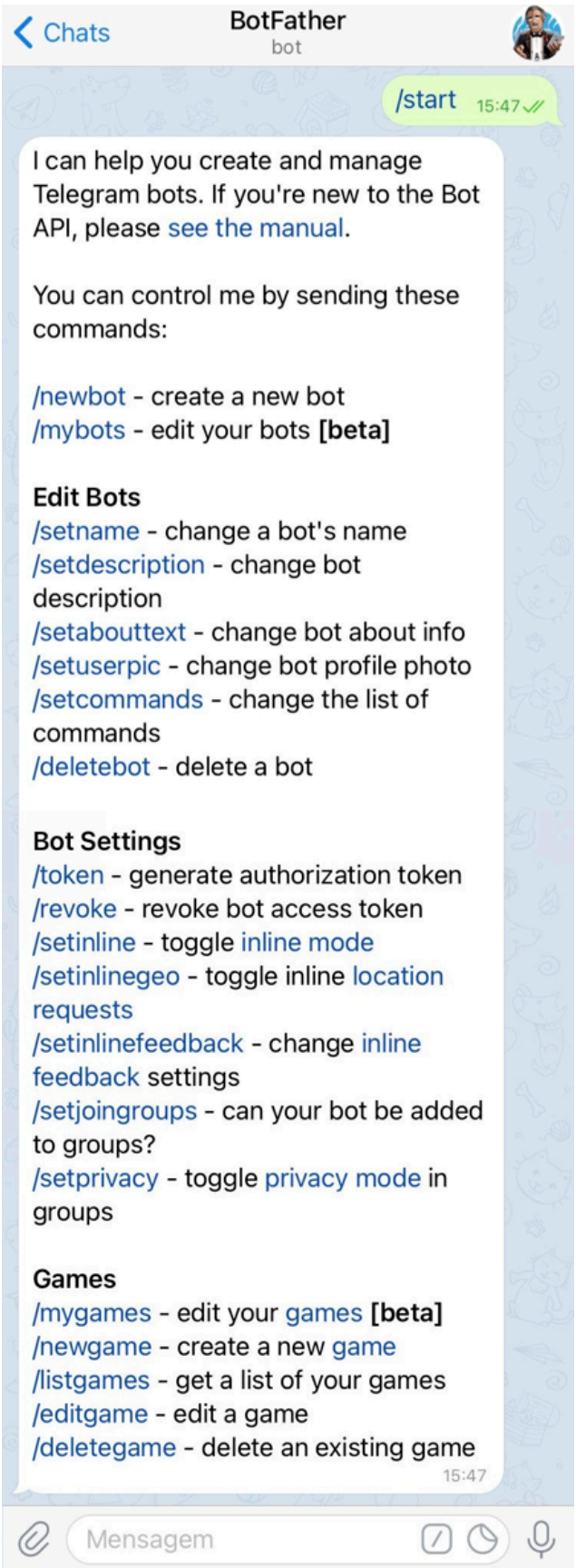

Chats
BotFather
bot
/start 15:47
I can help you create and manage Telegram bots. If you're new to the Bot API, please see the manual.
You can control me by sending these commands:
/newbot - create a new bot
/mybots - edit your bots [beta]
Edit Bots
/setname - change a bot's name
/setdescription - change bot description
/setabouttext - change bot about info
/setuserpic - change bot profile photo
/setcommands - change the list of commands
/deletebot - delete a bot
Bot Settings
/token - generate authorization token
/revoke - revoke bot access token
/setinline - toggle inline mode
/setinlinegeo - toggle inline location requests
/setinlinefeedback - change inline feedback settings
/setjoingroups - can your bot be added to groups?
/setprivacy - toggle privacy mode in groups
Games
/mygames - edit your games [beta]
/newgame - create a new game
/listgames - get a list of your games
/editgame - edit a game
/deletegame - delete an existing game
15:47
Mensagem

Figura 2.5: As opções de comando para criação e configuração de bots do Telegram

2.1 Colocando a mão na massa!

Agora que você já conhece todas as opções e configurações disponíveis, vamos finalmente criar nosso bot. Você pode digitar `/newbot` no bate-papo com o @BotFather ou clicar sobre a opção na mensagem de ajuda que ele mostra.

Na sequência, o @BotFather vai solicitar para você um nome para seu bot. Como você pode ver na imagem a seguir, escolhi para o meu bot o nome de `"meu_primeiro_pequeno_robo"` para evitar que já exista um com o mesmo nome, o que impossibilitaria sua criação.

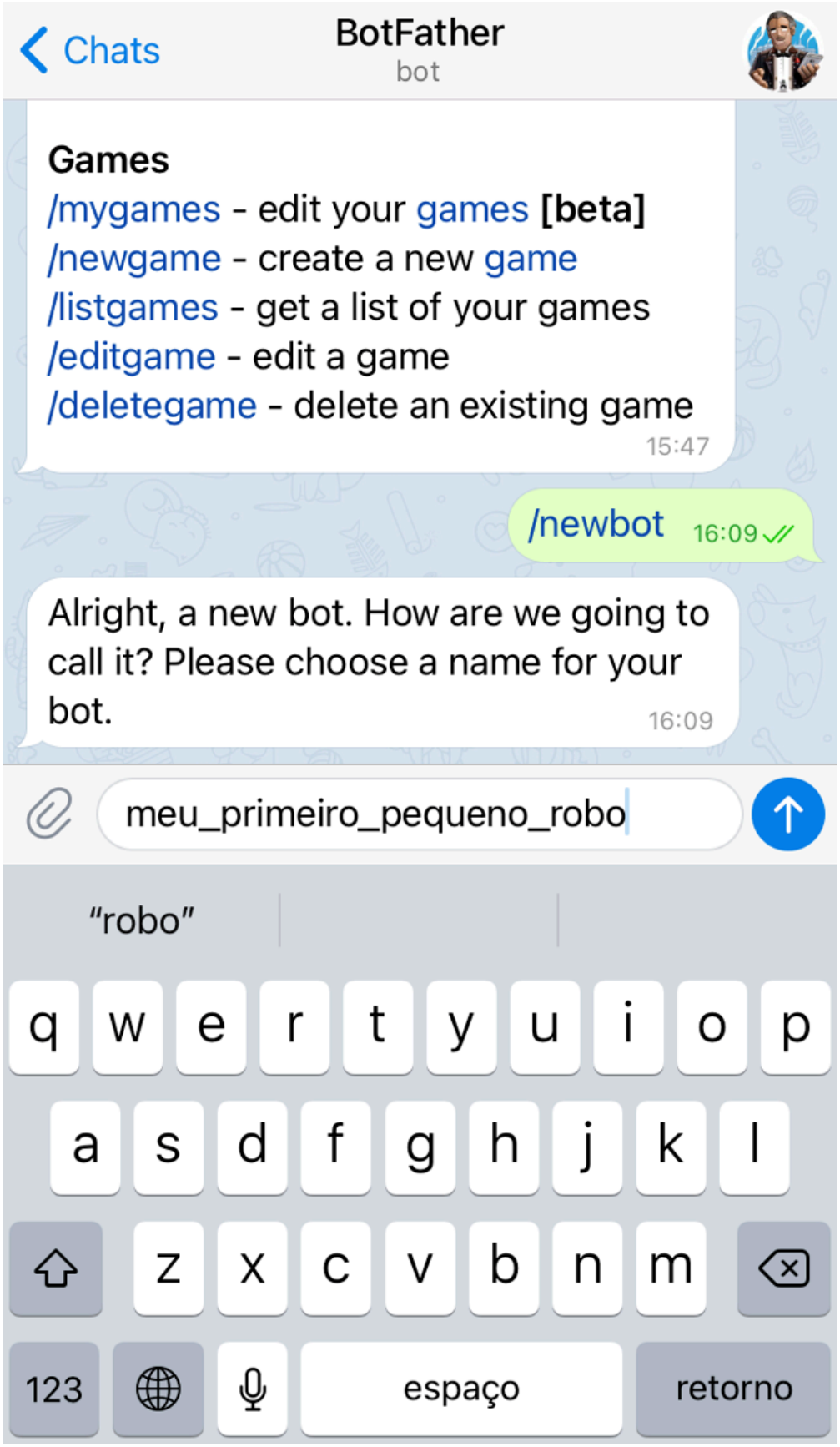

Chats
BotFather
bot
Games
/mygames - edit your games [beta]
/newgame - create a new game
/listgames - get a list of your games
/editgame - edit a game
/deletegame - delete an existing game
15:47
/newbot 16:09
Alright, a new bot. How are we going to call it? Please choose a name for your bot.
16:09
meu_primeiro_pequeno_robo
“robo”
q w e r t y u i o p
a s d f g h j k l
z x c v b n m
123
espaço
retorno

Figura 2.6: Opção /newbot e dando um nome para o bot

Logo depois de enviar o nome, o @BotFather solicita um nome de usuário (*username*) para o bot. Esse username deve, obrigatoriamente, terminar com "bot". Mesmo que não seja obrigatório, uma sugestão para que você sempre se lembre do nome de usuário do bot é usar o mesmo nome com a terminação "bot".

No meu caso, o nome do usuário do bot ficou `"meu_primeiro_pequeno_robo_bot"`, mas o nome de usuário do seu bot e o nome dele não precisam necessariamente ser iguais.

Chats
BotFather
bot
/deletegame - delete an existing game
15:47
/newbot 16:09
Alright, a new bot. How are we going to call it? Please choose a name for your bot.
16:09
meu_primeiro_pequeno_robo 16:11
Good. Now let's choose a username for your bot. It must end in `bot`. Like this, for example: TetrisBot or tetris_bot.
16:11
meu_primeiro_pequeno_robo_bot
“bot”
q w e r t y u i o p
a s d f g h j k l
z x c v b n m
123
espaço
retorno

Figura 2.7: Dando um nome de usuário para o bot criado

Não se preocupe com o tamanho do nome do bot ou do seu nome de usuário, isso apenas servirá para identificá-lo por ocasião de alteração das configurações. Na programação usamos apenas o token de acesso, que veremos logo à frente.

Assim que você enviar o nome de usuário do seu bot, o @BotFather vai responder com uma mensagem de congratulações com o token de acesso (*token to access*) do seu bot, como mostra a imagem a seguir. O token é usado para programarmos o bot, informando a conta em que ele deverá se conectar ao Telegram. Anote-o em algum lugar seguro, pois com ele é possível ter acesso às configurações, informações e controle do seu bot, além de poder conectar-se ao Telegram usando a conta dele.

É de extrema importância entender que o token dá acesso às configurações e todas as informações do bot, além de dar acesso ao bot propriamente dito. Dessa forma, se o token for descoberto e utilizado por alguém não autorizado, algo catastrófico pode acontecer.

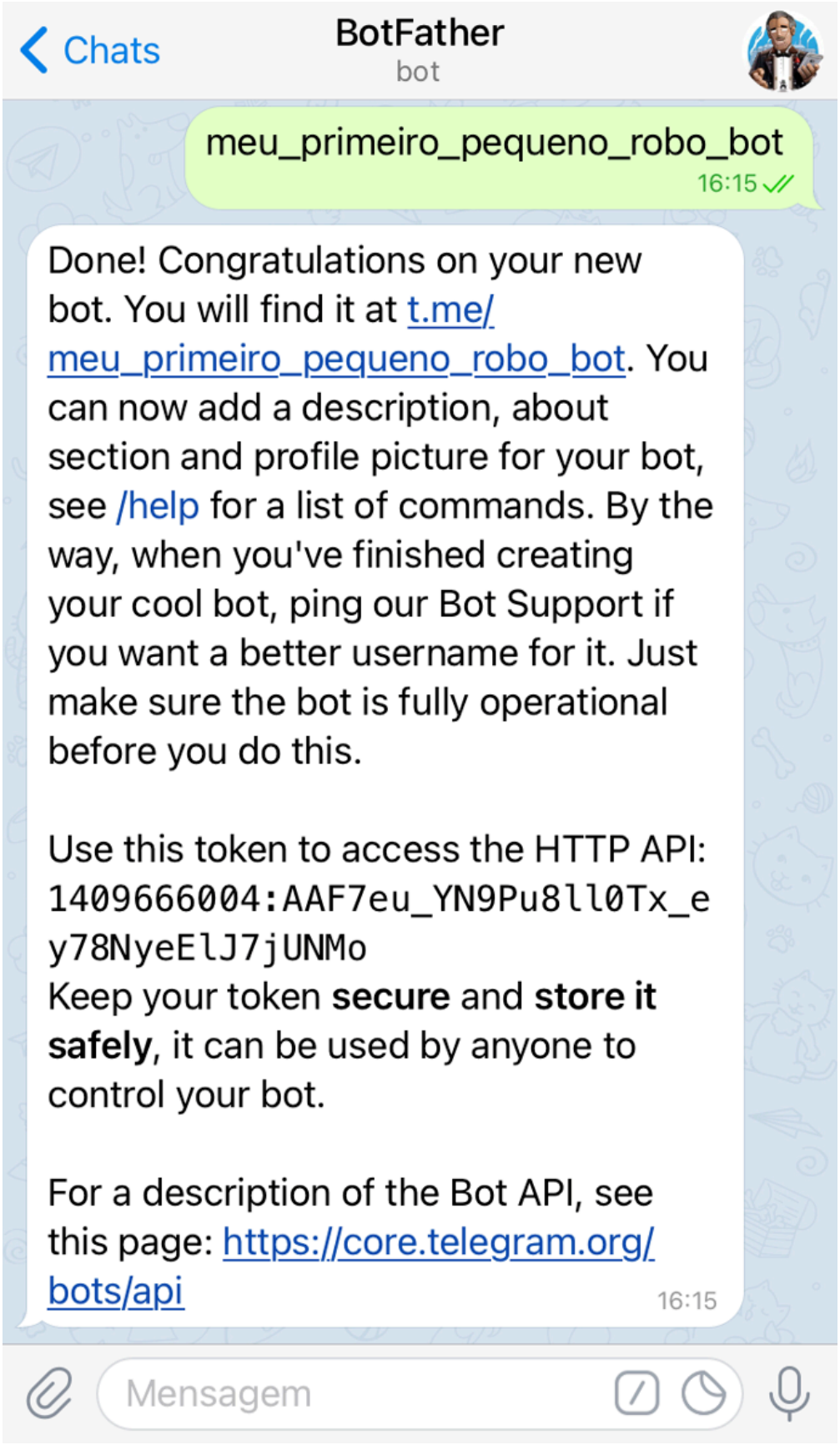

Chats
BotFather
bot
meu_primeiro_pequeno_robo_bot
16:15
Done! Congratulations on your new bot. You will find it at t.me/meu_primeiro_pequeno_robo_bot. You can now add a description, about section and profile picture for your bot, see /help for a list of commands. By the way, when you've finished creating your cool bot, ping our Bot Support if you want a better username for it. Just make sure the bot is fully operational before you do this.
Use this token to access the HTTP API:
1409666004:AAF7eu_YN9Pu8ll0Tx_ey78NyeElJ7jUNMo
Keep your token secure and store it safely, it can be used by anyone to control your bot.
For a description of the Bot API, see this page: https://core.telegram.org/bots/api
16:15
Mensagem

Figura 2.8: Mensagem de congratulações e com o token de acesso

O token exibido na imagem anterior foi utilizado somente como exemplo e foi revogado ao final da edição deste livro.

PARABÉNS! Cumprindo esses passos, você já tem o seu bot criado no Telegram. Antes de seguirmos para a programação dele, vamos realizar algumas configurações importantes.

Vamos escolher alguma imagem para o bot. No @BotFather, envie o comando `/setuserpic`. De imediato, o Telegram solicitará para que você selecione o nome do bot cuja imagem você quer ajustar, como na imagem a seguir.

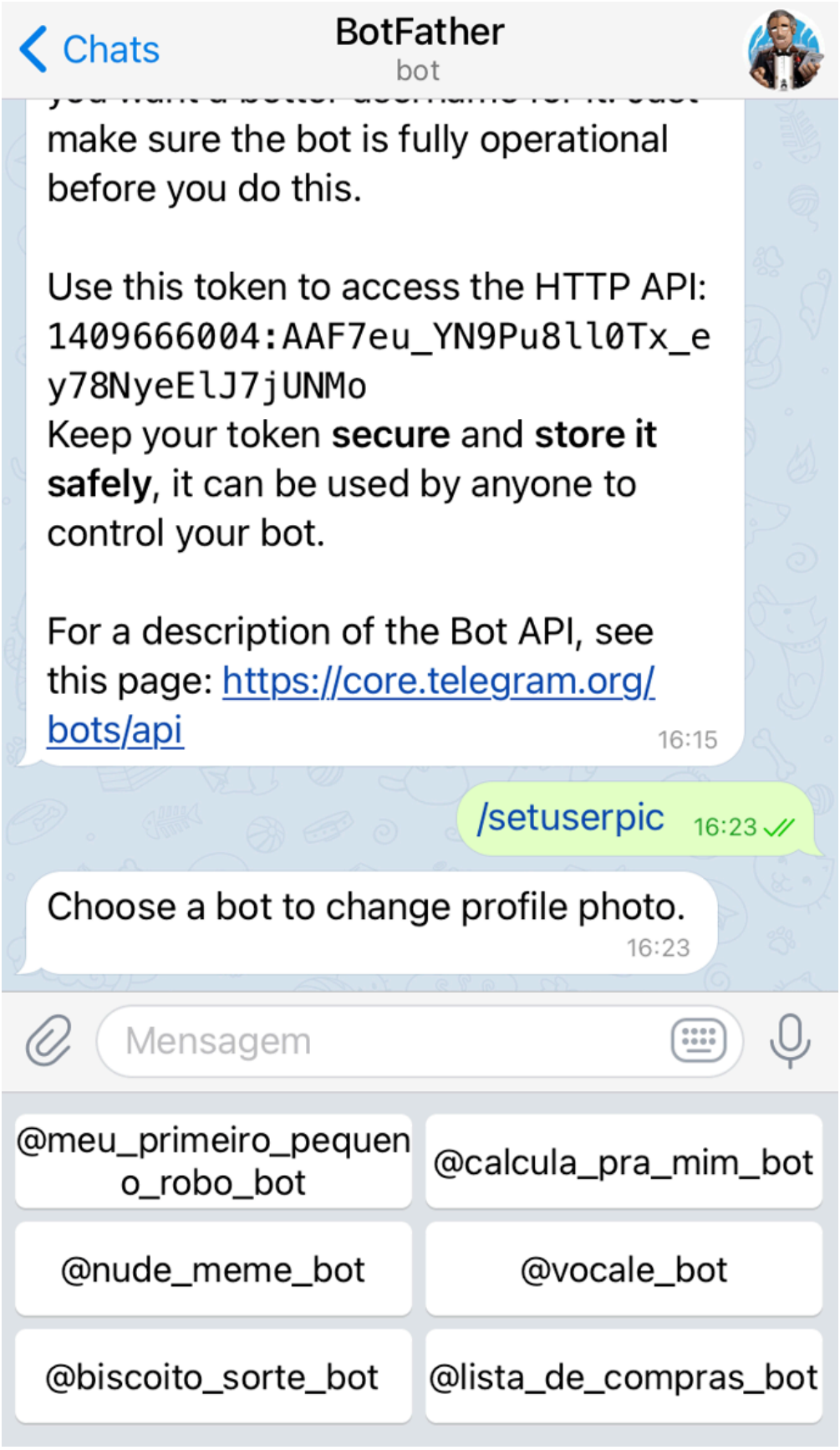

Chats
BotFather
bot
make sure the bot is fully operational before you do this.
Use this token to access the HTTP API:
1409666004:AAF7eu_YN9Pu8ll0Tx_ey78NyeElJ7jUNMo
Keep your token secure and store it safely, it can be used by anyone to control your bot.
For a description of the Bot API, see this page: https://core.telegram.org/bots/api
16:15
/setuserpic
16:23
Choose a bot to change profile photo.
16:23
Mensagem
@meu_primeiro_pequeno_robo_bot
@calcula_pra_mim_bot
@nude_meme_bot
@vocale_bot
@biscoito_sorte_bot
@lista_de_compras_bot

Figura 2.9: Selecionando o nome do bot para o qual enviaremos uma nova imagem

Agora vamos enviar uma imagem de perfil para o seu bot, que passará a ser mostrada quando qualquer pessoa acessá-lo para adicioná-lo a alguma conversa ou para interagir diretamente com ele, como acontece com qualquer usuário do Telegram. Você pode escolher qualquer imagem, a que mais lhe agradar, mas observe que a escolha da imagem será parte da identidade visual do seu bot e os usuários o identificarão e lembrarão dele a partir disso. Para o nosso exemplo, eu escolhi um desenho inédito e feito por mim mesmo à mão:

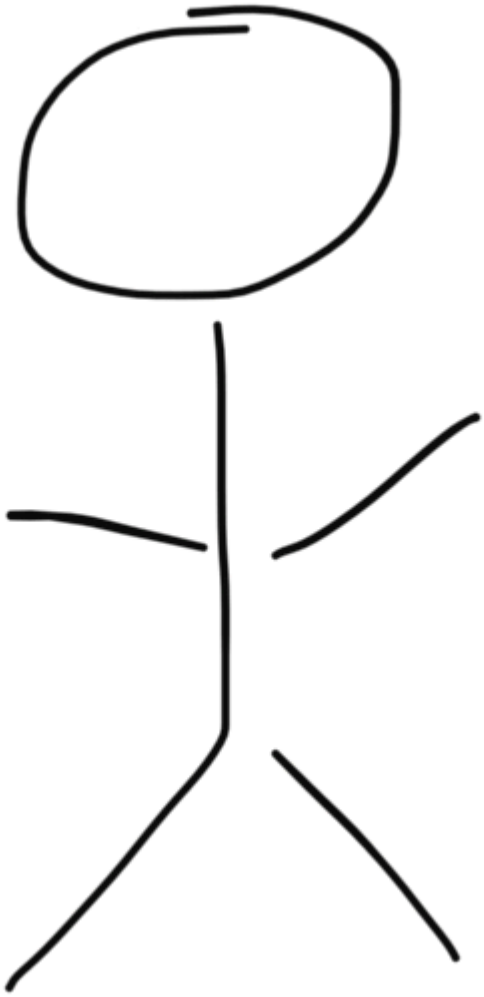


Figura 2.10: Minha obra prima!

Depois de enviar o arquivo, o @BotFather vai responder com uma mensagem de sucesso.

Agora que definimos a imagem do bot, acho importante ajustarmos algumas outras características: a descrição e o texto de sobre (`about`) do bot. Isso facilitará a vida dos usuários na hora de obter

informações sobre o funcionamento, a finalidade e a procedência do bot. Vamos fazer as duas configurações ainda no @BotFather.

Para ajustar a descrição, usamos o comando `/setdescription` e, depois, escolhemos qual robô estamos configurando. Eis o texto que escolhi para o meu bot: “Robô para bater papo exemplo do livro”.

Chats
BotFather
bot
@meu_primeiro_pequeno_robo_bot
13:45
OK. Send me the new description for the bot. People will see this description when they open a chat with your bot, in a block titled 'What can this bot do?'.
13:45
Robô para bater papo exemplo do livro
13:45
Success! Description updated. /help
13:45
Mensagem
Sim
Todo
I
Q W E R T Y U I O P
A S D F G H J K L
Z X C V B N M
123
space
return

Figura 2.11: Ajustando o texto de descrição do bot

Para ajustar o texto de sobre (`about`), usamos o comando `/setabouttext`. O texto que escolhi para o meu bot foi “Robô exemplo para ensinar as outras pessoas a fazerem bots”.

Chats
BotFather
bot
@meu_primeiro_pequeno_robo_bot
13:49
OK. Send me the new 'About' text. People will see this text on the bot's profile page and it will be sent together with a link to your bot when they share it with someone.
13:49
Robô exemplo para ensinar as outras pessoas a fazer bots
13:49
Success! About section updated. /help
13:49
Mensagem
Sim
Todo
I
Q W E R T Y U I O P
A S D F G H J K L
Z X C V B N M
123
space
return

Figura 2.12: Ajustes da descrição e do texto de sobre (about)

Por ora, acho que está bom, né!? Mais tarde sentiremos a necessidade de ajustar uma ou outra configuração – vamos mostrar como configurar e fazer outras sugestões enquanto colocamos a mão na massa.

Agora, vamos em frente porque chegou a hora de programar!

Capítulo 3

O primeiro bot – Olá, Mundo!

Chegou a tão esperada hora de dizer olá para o mundo com nosso primeiro chatbot!

Neste primeiro programa, vamos mostrar como fazer para rodá-lo tanto no Linux quanto no Windows. Primeiro no Linux, com as explicações de cada linha do código. Depois disso você saberá rodar seus bots e começará a compreender o funcionamento das linhas de código.

Considerando que você já tem tudo instalado e configurado conforme mostrado até agora, vamos à programação.

Rodar o bot no Linux

O nome do arquivo que utilizamos é `primeiro_bot.py`, mas você pode utilizar o nome que desejar, isso não influencia na programação. Apenas será necessário saber o nome dele para executarmos depois.

Para que o programa possa ser rodado diretamente do prompt de comandos do Linux sem a necessidade de utilizarmos nenhum outro comando adicional, é importante começarmos com as seguintes linhas:

```
#!/usr/bin/python3
#coding: utf-8
import telepot, time
```

A primeira linha, `#!/usr/bin/python3`, faz com que o arquivo seja reconhecido como um script Python3 e que possa ser executado diretamente pela linha de comando com `./primeiro_bot.py`. Caso contrário, você terá que especificar o ambiente Python para isso com um comando`python3 primeiro_bot.py`. Parece ser um pequeno

detalhe, mas na hora de agendar tarefas, por exemplo, esse é um atalho interessante.

Com `#coding: utf-8`, determinamos em qual esquema de caracteres estamos escrevendo. No caso, UTF-8 (8-bit Unicode Transformation Format) é o esquema padrão universal para Unicode compatível com ASCII e faz com que caracteres com acento do português brasileiro não sejam mostrados erroneamente.

Já a linha `import telepot, time` faz com que sejam importadas as bibliotecas Telepot, que já foi detalhada anteriormente, e a `time`, necessária para o funcionamento da Telepot. A biblioteca `time` possui funções para acesso e tratamento de datas e horários, ou seja, facilita na manipulação de acessos, formatações e conversões de tempo no Python.

Caso você esteja em uma rede corporativa ou por qualquer outro motivo acesse a internet por intermédio de um proxy, será necessário especificá-lo para que o bot possa receber e enviar dados pela internet. Podemos fazer isso com a seguinte linha:

```
telepot.api.set_proxy('http://endereço do proxy:porta do proxy',
('usuário','senha'))
```

Onde se lê `http://endereço do proxy:porta do proxy`, você deve especificar o endereço IP e a porta utilizada pelo proxy da sua rede (por exemplo, `http://192.168.0.1:3128`). Em seguida, você especifica o usuário e a senha que farão acesso através do proxy. Caso seu proxy não solicite usuário e senha, basta suprimir esses dois campos, especificando essa linha apenas com `telepot.api.set_proxy('http://endereço do proxy:porta do proxy')`.

Feitas as importações iniciais e tratado o detalhe do proxy (se necessário), vamos construir a função principal de nosso bot.

Nessa função, são tratados todos os comandos e mensagens que o bot receber e você pode nomeá-la como bem desejar. Essa função recebe como parâmetro a variável `msg`. Neste parâmetro, o Telepot enviará todas as informações sobre a mensagem (e, claro, a mensagem em si) para que possa ser decodificada e tratada da maneira correta.

```
def principal(msg):
        content_type, chat_type, chat_id = telepot.glance(msg)

        if content_type == 'text':
                chat_id = msg['chat']['id']
                mensagem = msg['text']

                if mensagem == 'Oi':
                        bot.sendMessage(chat_id,"Olá, Mundo!")
```

Os campos `content_type`, `chat_type` e `chat_id` são retornos do método `glance` da classe Telepot, que extrai as informações sobre a mensagem recebida.

O `content_type` refere-se ao tipo de conteúdo que foi recebido, que poderá ser uma das seguintes opções: `text` (texto); `audio` (áudio); `document` (documento); `game` (jogo); `photo` (fotografia); `sticker` (adesivo); `video` (vídeo); `voice` (voz); `video_note` (nota de vídeo); `contact` (contato); `location` (localização); `venue` (local); `new_chat_member` (novo membro de chat); `left_chat_member` (membro que deixou o chat); `new_chat_title` (novo título de chat); `new_chat_photo` (nova foto de chat); `delete_chat_photo` (exclusão de foto de chat); `group_chat_created` (criação de grupo de chat); `supergroup_chat_created` (criação de supergrupo de chat); `channel_chat_created` (criação de canal de chat); `migrate_to_chat_id` (migração para outra chat_id), `migrate_from_chat_id` (migração a partir de outra chat_id); `pinned_message` (mensagem fixada); `new_chat_members` (novos membros de chat); `invoice` (fatura); `successful_payment` (pagamento com sucesso).

É a entrada `content_type` que, basicamente, define como tratar o que está sendo recebido. Vemos isso já na linha seguinte, em `if content_type == 'text':`, que determina que se o tipo de conteúdo for um texto, executará as próximas linhas; caso contrário, não fará nada.

Dessa forma, o chatbot pode receber qualquer coisa do Telegram, mas reagirá apenas se o que for recebido for um texto, que ficará armazenado na variável mensagem a partir da linha `mensagem = msg['text'].`

Já na variável `chat_id` vamos armazenar a identificação do chat, que é única para cada conversa direta com usuário ou grupo em que o bot estiver inserido. Com a linha `chat_id = msg['chat']['id']` obtemos essa informação. Dessa forma, quando formos responder, vamos usar essa mesma identificação para que a mensagem de resposta seja enviada para o destinatário correto, ou seja, para o usuário ou o grupo correto.

Como armazenamos o texto da mensagem na variável mensagem, usamos o trecho:

```
if command == 'Oi':
                        bot.sendMessage(chat_id,"Olá, Mundo!")
```

Utilizamos essa estrutura para determinar se o que recebemos é um comando ou quaisquer outras coisas que desejamos processar. Nesse caso, se o texto da mensagem for igual a `"Oi"`, responderemos enviando uma mensagem `"Olá, Mundo!"` com a linha `bot.sendMessage(chat_id,"Olá, Mundo!").`

O método `sendMessage` necessita do `chat_id` especificado, para que a mensagem vá para o usuário ou grupo correto, e de um texto, que é a mensagem propriamente dita a ser enviada.

Terminada nossa função principal, é necessário apenas criar o objeto bot com a linha `bot =`

`telepot.Bot('1409666004:AAF7eu_YN9Pu8ll0Tx_ey78NyeElJ7jUNMo').`
Aqui, o método `Bot` da classe `telepot` recebe o token de acesso obtido na criação do bot no Telegram, como mostrado no capítulo anterior. **Lembre-se**: você precisa criar o seu próprio token e usá-lo. O token utilizado como exemplo neste livro já foi revogado.

Agora, informamos ao objeto `bot` que as mensagens devem ser enviadas para a função principal, onde serão tratadas. Fazemos isso com a seguinte linha:

```
bot.message_loop(principal)
```

Com isso, criamos um laço (*loop*) onde cada mensagem recebida deverá ser processada pela função principal.

Poderíamos terminar por aqui, mas aconteceria de o programa ser executado e terminar depois da última linha. Não é esse o comportamento que desejamos. Queremos que ele permaneça em funcionamento, receba as mensagens e processe-as quando determinado.

Para que o bot permaneça em funcionamento, criamos um loop infinito com o qual fazemos ele esperar 5 segundos antes de processar o que virá a seguir. Para essa instrução, utilizamos as seguintes linhas:

```
while 1:
        time.sleep(5)
```

Depois de tudo isso, o código-fonte do nosso primeiro bot ficará completo:

```
#!/usr/bin/python3
#coding: utf-8
import telepot, time

# telepot.api.set_proxy('http://192.168.0.1:3128',
('usuário','senha'))

def principal(msg):
```

```
        content_type, chat_type, chat_id = telepot.glance(msg)

        if content_type == 'text':
         chat_id = msg['chat']['id']
         mensagem = msg['text']

         if mensagem == 'Oi':
          bot.sendMessage(chat_id,"Olá, Mundo!")

bot =
telepot.Bot('1409666004:AAF7eu_YN9Pu8ll0Tx_ey78NyeElJ7jUNMo')
bot.message_loop(principal)

while 1:
        time.sleep(5)
```

Note que, no exemplo acima, a linha para uso do proxy está comentada, preenchida com nossos dados fictícios. Você pode descomentá-la e preenchê-la com os seus dados.

Finalmente, vamos executá-lo! Lembre-se de conceder a permissão antes:

```
chmod +x primeiro_bot.py
```

Agora, sim, é só executar:

```
./primeiro_bot.py
```

Se o programa ficar em funcionamento, sem mostrar nenhuma mensagem, principalmente de erro, é só ir ao Telegram e enviar mensagens para o bot.

Para isso, no Telegram, pesquise pelo nome do bot que você criou. No nosso exemplo, buscamos pelo nome do bot que criamos no capítulo anterior: `@meu_primeiro_pequeno_robo`.

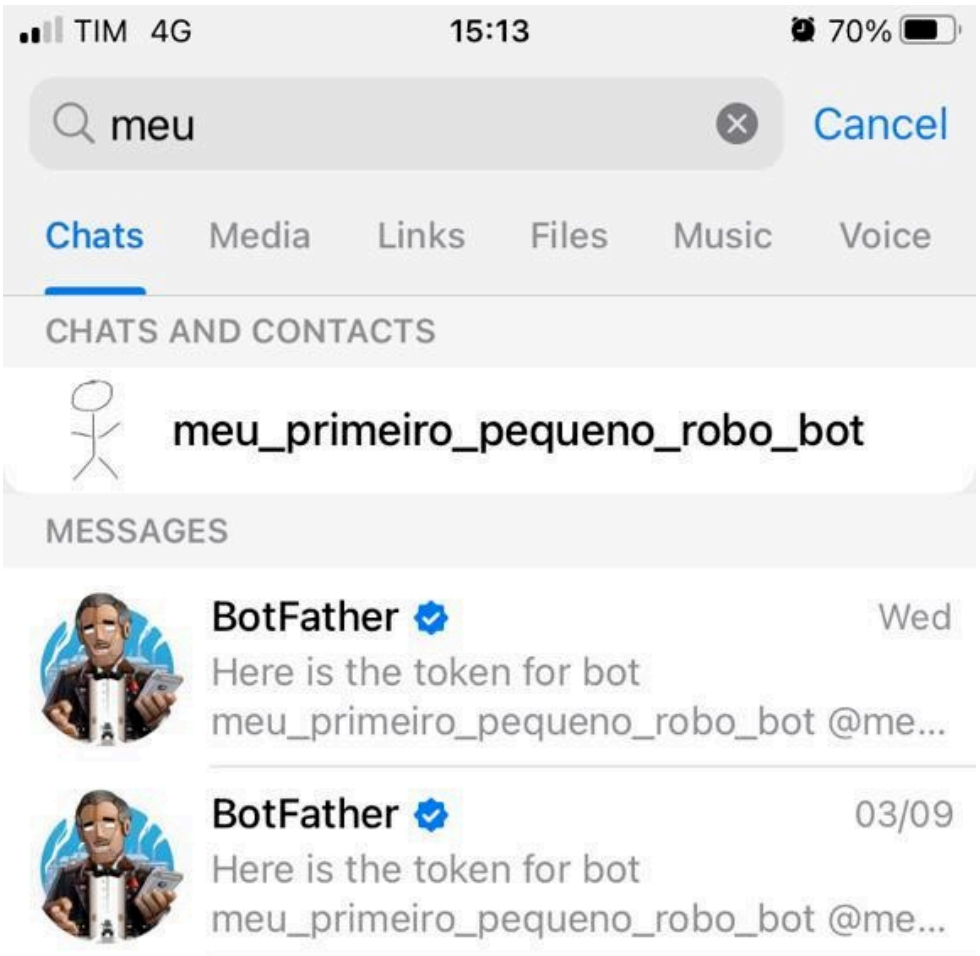

TIM 4G
15:13
70%
meu
Cancel
Chats
Media
Links
Files
Music
Voice
CHATS AND CONTACTS
meu_primeiro_pequeno_robo_bot
MESSAGES
BotFather
Wed
Here is the token for bot
meu_primeiro_pequeno_robo_bot @me...
BotFather
03/09
Here is the token for bot
meu_primeiro_pequeno_robo_bot @me...

Figura 3.1: Buscando o nosso primeiro bot

Toque ou clique nele. De imediato, ele abrirá o chat e mostrará a mensagem de descrição que definimos para ele.

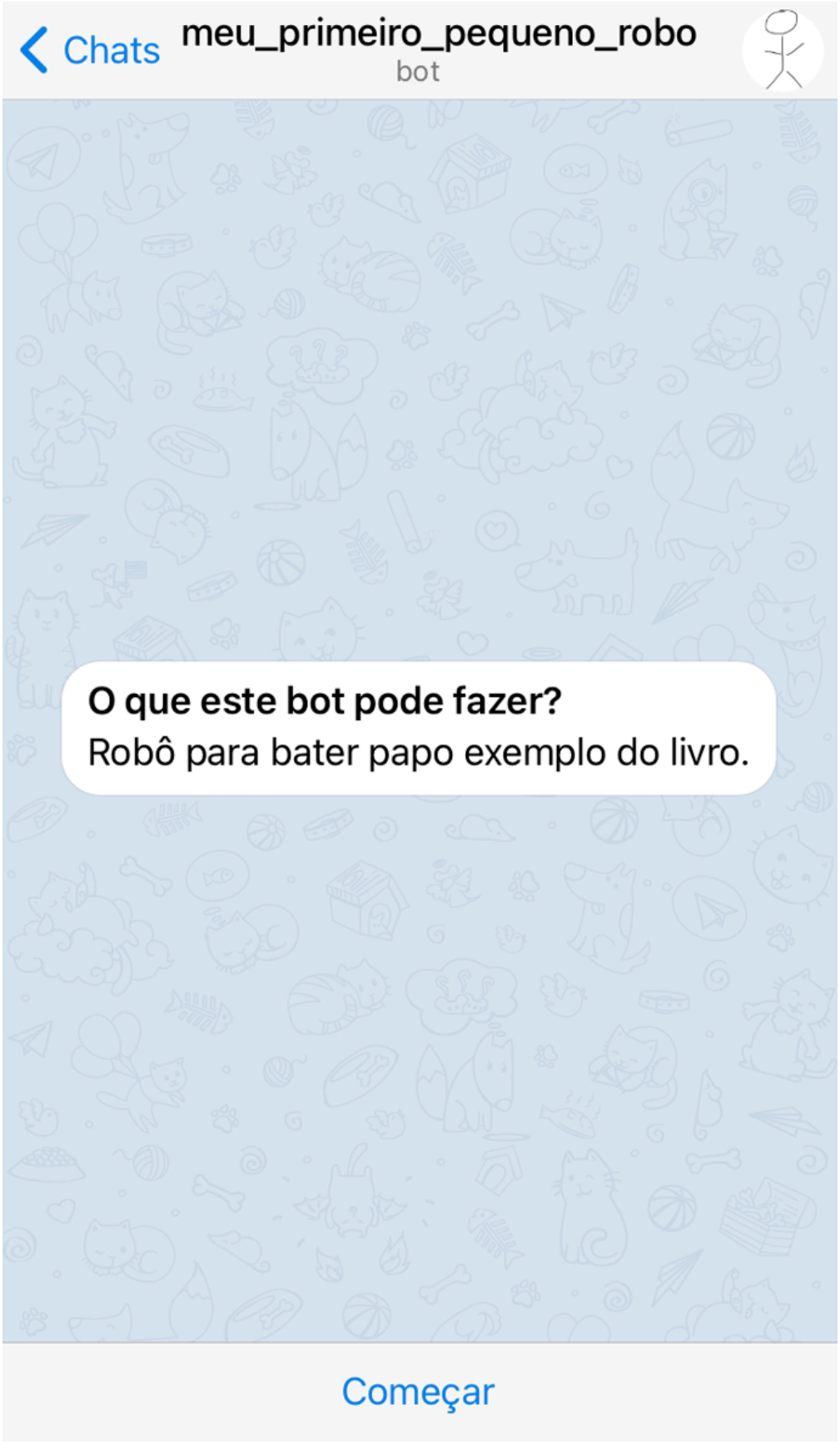

Chats
meu_primeiro_pequeno_robo
bot
O que este bot pode fazer?
Robô para bater papo exemplo do livro.
Começar

Figura 3.2: O chat com o bot e sua mensagem de descrição

Toque no botão “Começar” no rodapé da tela. Agora é só mandar mensagens para ele. Note que, caso o bot receba um “Oi”, ele responderá com “Olá, Mundo!”:

Chats
meu_primeiro_pequeno_robo
bot
O que este bot pode fazer?
Robô para bater papo exemplo do livro.
Hoje
/start 14:39
Teste 14:39
Oi 14:39
Hello World! 14:39
Mensagem
Sim
Todo
I
Q W E R T Y U I O P
A S D F G H J K L
Z X C V B N M
123
space
return

Figura 3.3: Enviando mensagens para o bot. Resposta para um “Oi”.

Perceba que, da maneira que está, se você enviar “oi” ou “OI” ele não responderá, pois na programação definimos apenas uma reposta para “Oi”, com a letra O maiúscula e a letra i minúscula.

Para que ele responda independentemente de como o “Oi” seja escrito, vamos alterar a linha `if mensagem == 'Oi'` para:

```
if mensagem.upper() == 'OI':
```

O método `upper()` converte a *string* contida na variável `mensagem` para que todas as letras fiquem em maiúsculas. Dessa forma, independente de como o “Oi” for enviado, ele será comparado com “OI”, com todas as letras maiúsculas.

Agora, é só rodar novamente o bot e testá-lo:

Chats
meu_primeiro_pequeno_robo
bot
O que este bot pode fazer?
Robô para bater papo exemplo do livro.
Hoje
/start 14:39
Teste 14:39
Oi 14:39
Hello World! 14:39
oi 14:43
Hello World! 14:43
Oi 14:43
Hello World! 14:43
ol 14:43
Hello World! 14:43
Ol 14:43
Hello World! 14:43
Mensagem

Figura 3.4: Enviando “Oi” de diversas formas

Funciona! Parabéns! Agora você já tem um chatbot para chamar de seu!

Rodar o bot no Windows

Para rodar o seu bot no Windows, abra o IDLE Python. Você verá o prompt de comandos dele, assim como vimos no capítulo 1 quando o instalamos e testamos.

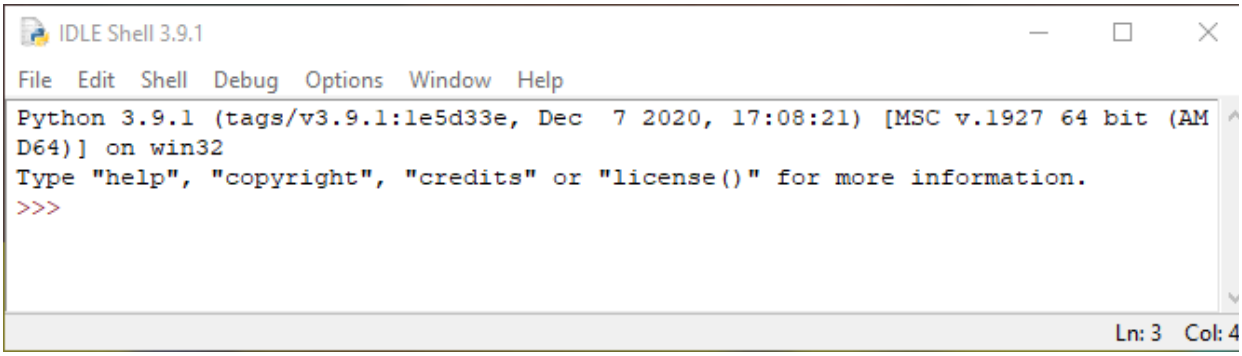


Figura 3.5: O IDLE Python no Windows

Vá ao menu File (Arquivo) e clique na opção New File (Novo Arquivo). Na nova janela, insira o código-fonte; a partir dela, poderemos salvá-lo em um arquivo e executá-lo.

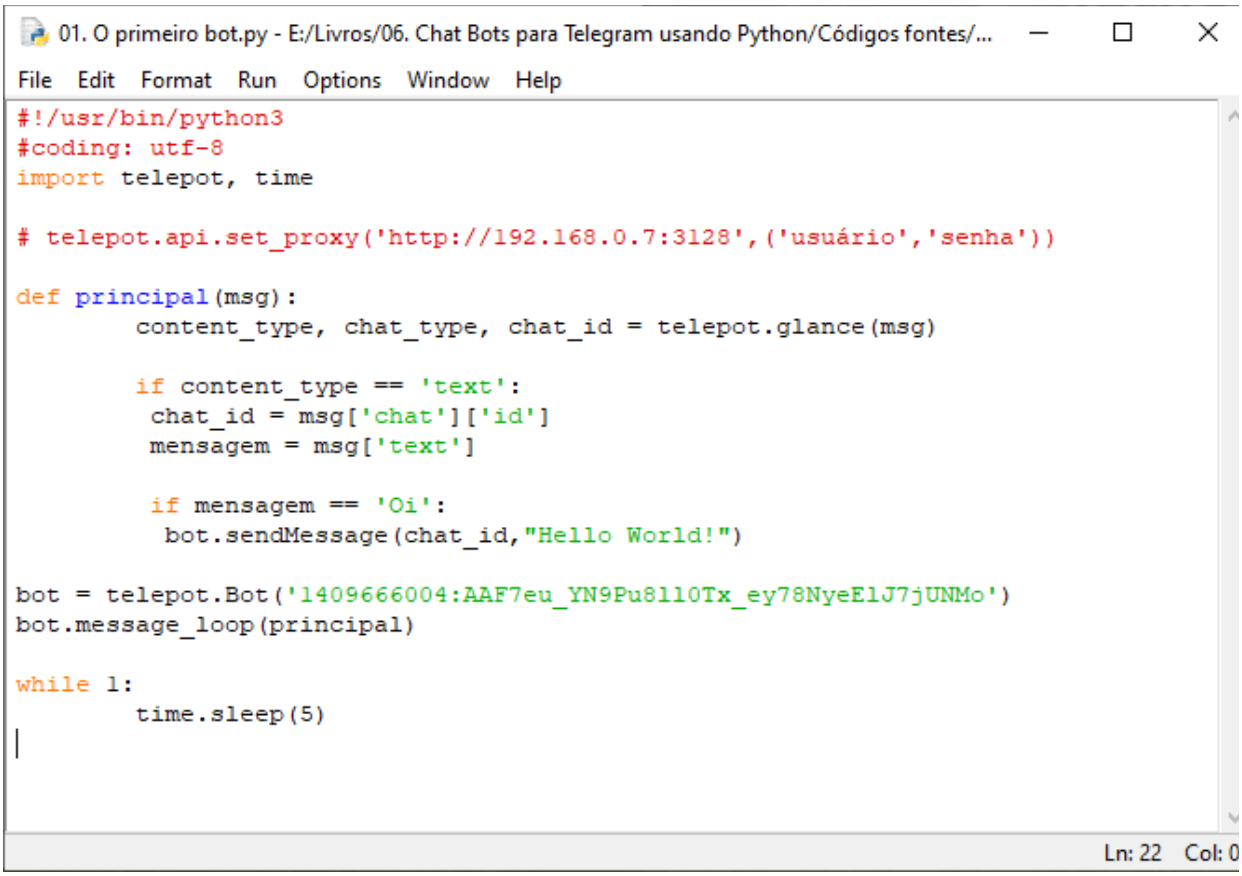


Figura 3.6: O código-fonte na janela de arquivo do IDLE Python

Para executar o código, é só ir ao menu `Run` (Executar) e clicar na opção `Run Module` (Executar Módulo) – ou pressionar a tecla de atalho `F5`.

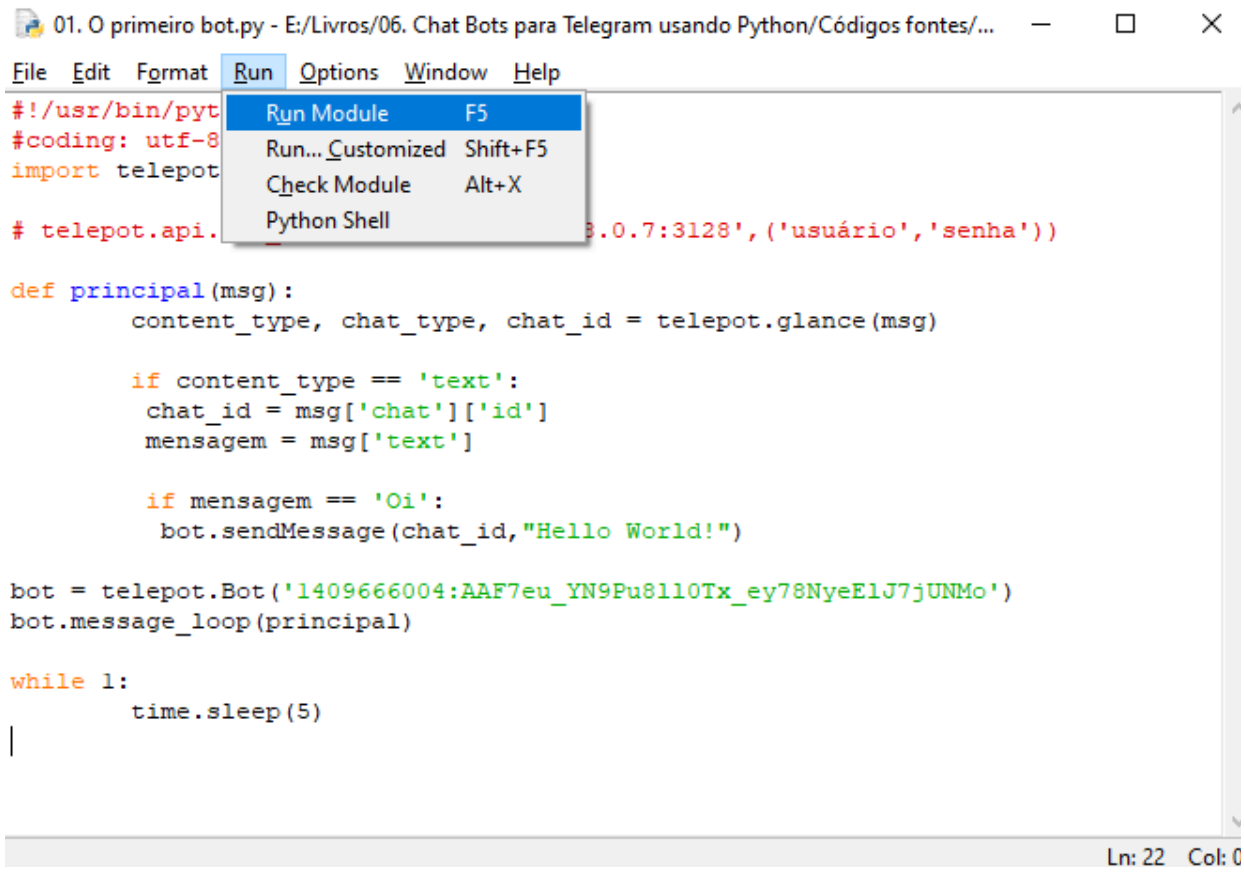


Figura 3.7: Executando o programa

Assim que colocar seu programa para executar, você verá na janela do IDLE Shell as saídas que ele mostra.

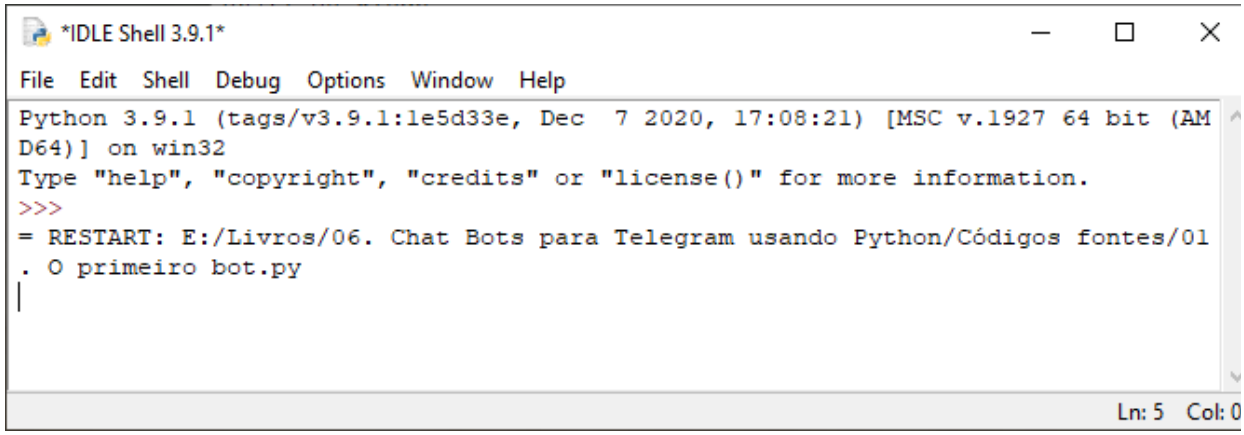


Figura 3.8: IDLE Shell com a execução do programa

Para interromper a execução, use as teclas de atalho CTRL + C ou o vá ao menu `Shell` e selecione a opção `Interrupt Execution` (Interromper Execução).

Finalmente, para verificar se tudo funciona, vá ao Telegram e envie um “Oi”.

Chats
meu_primeiro_pequeno_robo
bot
O que este bot pode fazer?
Robô para bater papo exemplo do livro.
Hoje
/start 14:39
Teste 14:39
Oi 14:39
Hello World! 14:39
Mensagem
Sim
Todo
I
Q W E R T Y U I O P
A S D F G H J K L
Z X C V B N M
123
space
return

Figura 3.9: Resposta para um “Oi” no chatbot

Lembre-se do problema do Python em reconhecer letras maiúsculas e minúsculas como diferentes. Basta usar o método `upper()` para converter toda a *string* em maiúsculas e teremos a solução.

```
if mensagem.upper() == 'OI':
```

Rode novamente o chatbot para testá-lo:

Chats
meu_primeiro_pequeno_robo
bot
O que este bot pode fazer?
Robô para bater papo exemplo do livro.
Hoje
/start 14:39
Teste 14:39
Oi 14:39
Hello World! 14:39
oi 14:43
Hello World! 14:43
Oi 14:43
Hello World! 14:43
ol 14:43
Hello World! 14:43
Ol 14:43
Hello World! 14:43
Mensagem

Figura 3.10: Enviando “Oi” de diversas formas

Se você testou e não apareceu nenhuma mensagem de erro, parabéns! Nosso bot funciona e responde corretamente.

Vamos em frente, mas com calma, pois ainda temos muito a aprender! No próximo capítulo, vamos saber como programar um bot para ler dados da internet e fornecer a previsão do tempo.

Capítulo 4

Lendo dados da internet - um bot para previsão do tempo

Continuando com a ideia de que o bot receberá comandos de texto, como o nosso primeiro bot do capítulo anterior, o desafio deste capítulo é conseguir a previsão do tempo diretamente do INMET, o Instituto Nacional de Meteorologia.

Esse bot servirá para muitas coisas além de conseguir a previsão do tempo. Com ele, será possível ver como fazer requisições HTTP usando Python, como tratar um arquivo no formato JSON e também como fazer algumas operações com data.

A cada exemplo, a cada bot que faremos, teremos um novo desafio, que vai muito além do processar textos e comandos enviados via Telegram. Vamos realizar processamentos por vezes complexos e aprofundar na linguagem Python.

Para nosso bot de previsão de tempo, utilizaremos o site do INMET (inmet.gov.br), que disponibiliza vários dados meteorológicos através de APIs que retornam dados no formato JSON. Nosso desafio para esse bot é receber um comando de texto (por exemplo, "clima"), acessar o INMET e ler os dados da API de previsão de tempo.

Para obter os dados de previsão de tempo, basta acessar a URL https://apiprevmet3.inmet.gov.br/previsao/(código da cidade), onde `código da cidade` é um código em número fornecido pelo IBGE para cada cidade. A lista com os códigos do IBGE para cada cidade pode ser obtida em: https://www.ibge.gov.br/explica/codigos-dos-municipios.php. Busque pela sua cidade e faça um teste!

Vamos testar, por exemplo, a previsão do tempo para Brasília/DF. No site do IBGE, o código da cidade é 5300108. Troque o código da sua cidade (se você fez o teste indicado anteriormente) pelo código de Brasília. A URL ficará assim: https://apiprevmet3.inmet.gov.br/previsao/5300108.

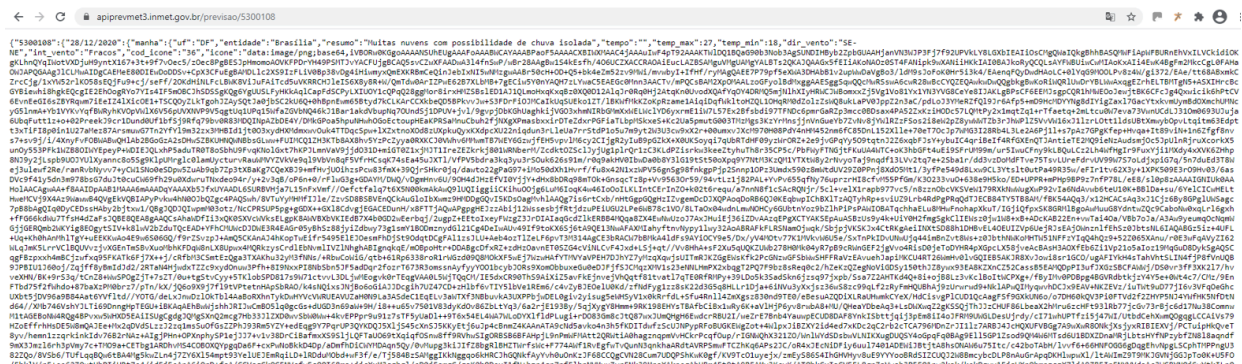


Figura 4.1: Retorno da API de previsão do tempo do INMET para Brasília/DF {w100%}

Os dados são sempre de dia atual e para os próximos quatro dias. Os dois primeiros dias trazem a previsão do tempo para os períodos da manhã, tarde e noite; já os demais, apenas a previsão do tempo geral. O resultado tem uma sequência estrutura (na frente de cada campo, segue um exemplo de como pode vir o respectivo campo):

Campo - Conteúdo - Descrição

- `"5300108":` - Código da cidade fornecido pelo IBGE
- `{`
- `"01/01/2021":` - Data atual
- `{`|-
- `"manha":` - Período

- "uf":"DF", - UF da cidade
- "entidade":"Brasília", - Nome da cidade
- "resumo":"Muitas nuvens com chuva", - Resumo da previsão do tempo
- "tempo":"", - Tempo atual
- "temp_max":27, - Temperatura máxima para o período
- "temp_min":18, - Temperatura mínima para o período
- "dir_vento":"SE-NE", - Direção do vento
- "int_vento":"Moderado", - Intensidade do vento
- "cod_icone":"36", - Código do ícone da previsão do tempo
- "ícone":"data:image/png;base64", - Imagem do ícone em PNG (com codificação 64 bits)
- "dia_semana":"Segunda-feira", - Dia da semana por extenso
- "umidade_max":95, - Umidade máxima no período
- "umidade_min":45, - Umidade mínima no período
- "temp_max_tende":"Estável", - Tendência da temperatura máxima
- "cod_temp_max_tende_icone":"74", - Código do ícone da previsão do tempo
- "temp_max_tende_icone" "data:image/png;base64", - Imagem do ícone em PNG (com codificação 64 bits)
- "estacao":"Verão", - Nome da estação do ano
- "hora":"16", - Hora da última atualização
- "nascer":"05h41", - Horário do nascer do Sol
- "ocaso":"18h45", - Horário do pôr do Sol
- "fonte":"prevmet" - Fonte dos dados
- }

Ainda nesse exemplo, os dados prosseguem com os períodos da tarde e noite para os dois primeiros dias. Vemos, a seguir, os campos para esses períodos e seus respectivos conteúdos:

- "tarde":
- {
- "uf":"DF",

- "entidade":"Brasília",
- "resumo":"Muitas nuvens com chuva",
- "tempo":"",
- "temp_max":27,
- "temp_min":18,
- "dir_vento":"SE-NE",
- "int_vento":"Moderado",
- "cod_icone":"36",
- "ícone":"data:image/png;base64",
- "dia_semana":"Segunda-feira",
- "umidade_max":95,
- "umidade_min":45,
- "temp_max_tende":"Estável",
- "cod_temp_max_tende_icone":"74",
- "temp_max_tende_icone" "data:image/png;base64",
- "estacao":"Verão",
- "hora":"16",
- "nascer":"05h41",
- "ocaso":"18h45",
- "fonte":"prevmet"
- }
- "noite":
- {
- "uf":"DF",
- "entidade":"Brasília",
- "resumo":"Muitas nuvens com chuva",
- "tempo":"",
- "temp_max":27,
- "temp_min":18,
- "dir_vento":"SE-NE",
- "int_vento":"Moderado",
- "cod_icone":"36",
- "ícone":"data:image/png;base64",
- "dia_semana":"Segunda-feira",

- "umidade_max":95,
- "umidade_min":45,
- "temp_max_tende":"Estável",
- "cod_temp_max_tende_icone":"74",
- "temp_max_tende_icone" "data:image/png;base64",
- "estacao":"Verão",
- "hora":"16",
- "nascer":"05h41",
- "ocaso":"18h45",
- "fonte":"prevmet"
- }
- }

O mesmo se repete para a próxima data. Para as três últimas datas da previsão do tempo, sem os períodos para manhã, tarde e noite, a estrutura é a seguinte (também com exemplos de preenchimento):

- "03/01/2021":
- {
- "uf":"DF",
- "entidade":"Brasília",
- "resumo":"Muitas nuvens com chuva",
- "tempo":"",
- "temp_max":27,
- "temp_min":18,
- "dir_vento":"SE-NE",
- "int_vento":"Moderado",
- "cod_icone":"36",
- "ícone":"data:image/png;base64",
- "dia_semana":"Segunda-feira",
- "umidade_max":95,
- "umidade_min":45,
- "temp_max_tende":"Estável",
- "cod_temp_max_tende_icone":"74",
- "temp_max_tende_icone" "data:image/png;base64",

- "estacao":"Verão",
- "hora":"16",
- "nascer":"05h41",
- "ocaso":"18h45",
- "fonte":"prevmet"
- }

Enfim, poderíamos ficar destrinchando apenas esse JSON do INMET durante todo o restante do livro, mas já temos as informações básicas para continuarmos: a estrutura básica do que o Instituto Nacional de Meteorologia nos retorna para a previsão do tempo em uma determinada localidade.

Já de início precisaremos de uma nova biblioteca para conseguir ler o arquivo JSON da URL do INMET. Essa biblioteca é a *Requests*. Ela faz com que as requisições do tipo HTTP sejam fáceis e possam ser escritas em poucas linhas.

Resumindo a instalação:

No Windows, use o seguinte comando:

pip install requests

No Linux, use o seguinte comando:

pip3 install requests

Instalada a biblioteca Requests, começaremos nosso programa com:

```
#!/usr/bin/python3
#coding: utf-8
import telepot, time, json, requests
from datetime import date, timedelta
```

As bibliotecas Telepot e Time são necessárias para o funcionamento do bot, para que ele possa enviar e receber as mensagens do Telegram.

A biblioteca JSON tem o mesmo espírito da Requests mencionada acima. A ideia é prover objetos e métodos que facilitem o processamento de arquivos no formato JSON.

Com `from datetime import date, timedelta`, importamos apenas as classes `date` e `timedelta` da biblioteca `datetime`. Não fazemos simplesmente `import datetime` porque essa biblioteca é enorme; importá-la completamente só faria com que o programa final fosse muito grande, além de perder desempenho. Você pode fazer isso com todas as bibliotecas que desejar: importar somente as classes que utilizará no seu programa para que ele fique menor e mais rápido!

A biblioteca `datetime` traz várias classes e métodos para obtenção, manipulação e formatação de data e horas no Python. A classe `date` traz essas facilidades para datas e `timedelta` traz facilidades para a manipulação de datas, tais como soma, diferença e cálculo de períodos, sejam datas ou horas.

Continuamos nosso programa com o seguinte bloco:

```
# telepot.api.set_proxy('http://192.168.0.1:3128',
('usuario','senha'))

# proxy =   {
#   'http': 'http://usuário:senha@192.168.0.1:3128',
#   'https': 'http://usuário:senha@192.168.0.1:3128',
# }
```

Note que de início usei o caractere `#` para que essas linhas não sejam executadas. Elas estarão como comentários, pois são as configurações de proxy que precisam ser utilizadas caso você esteja utilizando um em sua rede.

É interessante ter esse código pronto e apenas comentado, pois, caso seja necessário, será muito mais prático e conveniente apenas descomentá-lo, uma vez que já está "na mão".

Vamos detalhar o trecho onde o bot obtém a previsão do tempo e a envia de volta ao usuário, que a solicitou ao enviar a palavra "clima" via Telegram. Note que o início do código já foi tratado anteriormente. A função principal será assim:

```
def principal(msg):
    content_type, chat_type, chat_id = telepot.glance(msg)
    if content_type == 'text':
        chat_id = msg['chat']['id']
        mensagem = msg['text']
        if mensagem.upper() == 'CLIMA':
            requisicao =
requests.get("https://apiprevmet3.inmet.gov.br/previsao/5300108"
,proxies=proxy).json()
            data = date.today()
            resposta = "O clima para " + requisicao["5300108"]
[str(data.strftime("%d/%m/%Y"))]["manha"]["entidade"] + "\n"
            for i in range(2):
                dataf = data.strftime("%d/%m/%Y")
                resposta += dataf + "\nmanhã - " +
requisicao["5300108"][str(dataf)]["manha"]["resumo"] + "\n"
                resposta += "tarde - " + requisicao["5300108"]
[str(dataf)]["tarde"]["resumo"] + "\n"
                resposta += "noite - " + requisicao["5300108"]
[str(dataf)]["noite"]["resumo"] + "\n"
                data = data + timedelta(days = +1)
                for i in range(3):
                    dataf = data.strftime("%d/%m/%Y")
                    resposta += dataf + " - " +
requisicao["5300108"][str(dataf)]["resumo"] + "\n"
                    data = data + timedelta(days = +1)
                    bot.sendMessage(chat_id,resposta)
```

É no trecho dentro da estrutura de decisão `if mensagem.upper() == 'CLIMA':` que toda a mágica acontece: conseguir a previsão do tempo, obter os dados que nos interessam e enviar de volta ao solicitante.

Com a linha `requisicao = requests.get("https://apiprevmet3.inmet.gov.br/previsao/5300108" ,proxies=proxy).json()`, acessamos (`get`) a URL https://apiprevmet3.inmet.gov.br/previsao/5300108 e armazenamos

o conteúdo dela na variável `requisicao`. Lembre-se que 5300108 é o código da cidade (Brasília, no caso) obtido no site do IBGE.

A opção `proxies=proxy` pode ser omitida se a sua rede não possui proxy para filtragem de conteúdo.

Indicamos que o conteúdo obtido pela requisição HTTP está no formato JSON quando colocamos o método `.json()` no final. Com isso, a variável `requisicao` terá o seu conteúdo já formatado como JSON, o que facilita a busca e o tratamento das chaves.

Ao invés de enviarmos várias mensagens para o Telegram a cada novo dado gerado, acumulamos tudo o que queremos enviar na variável `resposta` e, apenas no final de todo o processamento, enviamos de volta para o solicitante via Telegram. Isso faz com que o programa tenha um desempenho maior, já que a cada envio de mensagem é necessário um novo acesso à internet, o que, mesmo com a rede mais rápida possível, leva o tempo necessário para a transmissão multiplicado pela quantidade de mensagens enviadas.

Com a linha `resposta = "O clima para " + requisicao["5300108"][str(date.today().strftime("%d/%m/%Y"))]["manha"]["entidade"] + "\n"`, fazemos várias mágicas de uma só vez. Vamos entender melhor.

Como `requisicao` já está no formato JSON, acessamos a primeira chave do arquivo obtido do INMET, que é o código da cidade, com o trecho `requisicao["5300108"]`. Com `data = date.today()`, guardamos na variável `data` a data atual do sistema. Com o trecho `[str(data.strftime("%d/%m/%Y"))]`, formamos a segunda chave do JSON, que é a data da primeira previsão do tempo, no caso a data atual no formato DD/MM/YYYY.

Como o método `today()` da classe date retorna a data no formato YYYY-MM-DD, usamos o método `srtftime` para deixá-la no formato

esperado para essa chave JSON. No final de tudo, temos que converter essa data em string com a função `str`.

A partir daí temos dois trechos diferentes para processarmos os dados do clima. Das cinco datas retornadas, as duas primeiras possuem previsão por período (manhã, tarde e noite) e as três últimas, apenas o resumo geral da previsão.

Processamos o primeiro trecho com um laço de repetição em `for i in range(2):`, que fará com que o bloco seja processado duas vezes, incrementando a data.

```
dataf = data.strftime("%d/%m/%Y")
resposta += dataf + "\nmanhã - " + requisicao["5300108"]
[str(dataf)]["manha"]["resumo"] + "\n"
resposta += "tarde - " + requisicao["5300108"][str(dataf)]
["tarde"]["resumo"] + "\n"
resposta += "noite - " + requisicao["5300108"][str(dataf)]
["noite"]["resumo"] + "\n"
data = data + timedelta(days = +1)
```

Na variável `dataf = data.strftime("%d/%m/%Y")`, formatamos a data atual para que fique no padrão brasileiro, de forma a ser mais inteligível.

Nas três linhas seguintes formamos a string de resposta com a mensagem que será enviada pelo Telegram. Cada uma possuirá a previsão do tempo no período, que poderá ser `["manha"]`, `["tarde"]` ou `["noite"]`, definido pela terceira chave JSON. Os caracteres `\n` inserem uma quebra de linha.

Com a linha `data = data + timedelta(days = +1)`, incrementamos a data em um dia — ou seja, a cada laço, a data avançará, iniciando na data atual do sistema. Com isso, teremos os dois primeiros dias da previsão do tempo com os períodos de cada um.

O próximo laço de repetição, `for i in range(3):`, faz com que o bloco de comandos seja repetido para os próximos 3 dias da previsão do tempo.

```
dataf = data.strftime("%d/%m/%Y")
resposta += dataf + " - " + requisicao["5300108"][str(dataf)]
["resumo"] + "\n"
data = data + timedelta(days = +1)
```

Isso é feito exatamente como no trecho anterior, incrementando a data a cada laço, mas a string de resposta não terá a chave JSON correspondente ao período (manhã, tarde e noite), apenas o resumo.

Para finalizar, precisamos criar o objeto Bot com o token de acesso. Neste livro usaremos sempre o mesmo token, criado no início da jornada:

```
bot =
telepot.Bot('1409666004:AAF7eu_YN9Pu8ll0Tx_ey78NyeElJ7jUNMo')
```

Também temos que indicar qual será a função principal que processará todas as mensagens recebidas:

```
bot.message_loop(principal)
```

E, finalmente, fazer com que nosso programa não pare de funcionar:

```
while 1:
    time.sleep(5)
```

Com tudo isso, temos o código-fonte completo deste bot que, quando recebe a mensagem "clima" (Clima, cLima, climA ou qualquer outra variação de maiúsculas e minúsculas), retorna a previsão do tempo para os próximos 5 dias obtida do INMET.

O código completo fica assim:

```
#!/usr/bin/python3
#coding: utf-8
import telepot, time, json, requests
from datetime import date, timedelta

#telepot.api.set_proxy('http://192.168.0.1:3128',
('usuário','senha'))

# proxy =   {
```

```
    # 'http': 'http://d1396:p9d7s8v7@192.168.0.1:3128',
#   'https': 'http://d1396:p9d7s8v7@192.168.0.1:3128',
# }

def principal(msg):
    content_type, chat_type, chat_id = telepot.glance(msg)

    if content_type == 'text':
        chat_id = msg['chat']['id']
        mensagem = msg['text']

        if mensagem.upper() == 'CLIMA':
            requisicao =
requests.get("https://apiprevmet3.inmet.gov.br/previsao/5300108"
,proxies=proxy).json()
            data = date.today()
            resposta = "O clima para " + requisicao["5300108"]
[str(data.strftime("%d/%m/%Y"))]["manha"]["entidade"] + "\n"
            for i in range(2):
                dataf = data.strftime("%d/%m/%Y")
                resposta += dataf + "\nmanhã - " +
requisicao["5300108"][str(dataf)]["manha"]["resumo"] + "\n"
                resposta += "tarde - " + requisicao["5300108"]
[str(dataf)]["tarde"]["resumo"] + "\n"
                resposta += "noite - " + requisicao["5300108"]
[str(dataf)]["noite"]["resumo"] + "\n"
                data = data + timedelta(days = +1)
                for i in range(3):
                    dataf = data.strftime("%d/%m/%Y")
                    resposta += dataf + " - " +
requisicao["5300108"][str(dataf)]["resumo"] + "\n"
                    data = data + timedelta(days = +1)
                    bot.sendMessage(chat_id,resposta)

bot =
telepot.Bot('1409666004:AAF7eu_YN9Pu8ll0Tx_ey78NyeElJ7jUNMo')
bot.message_loop(principal)

while 1:
    time.sleep(5)
```

Agora é só colocá-lo para funcionar. Lembra-se de como faz? Tecla F5 ou menu Run (Executar), opção Run Module (Executar Módulo):

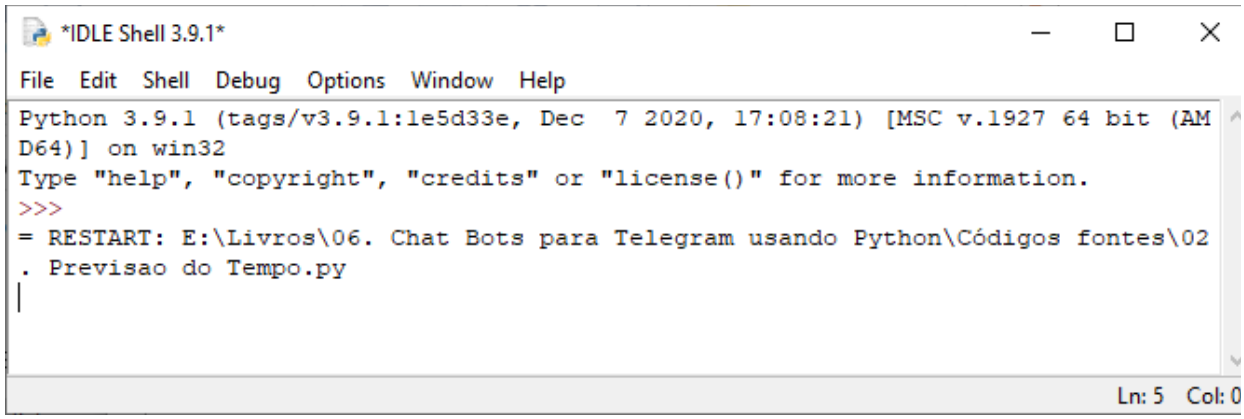


Figura 4.2: IDLE Shell indicando o funcionamento do bot

Depois, vá ao Telegram e envie a palavra "clima" para o bot.

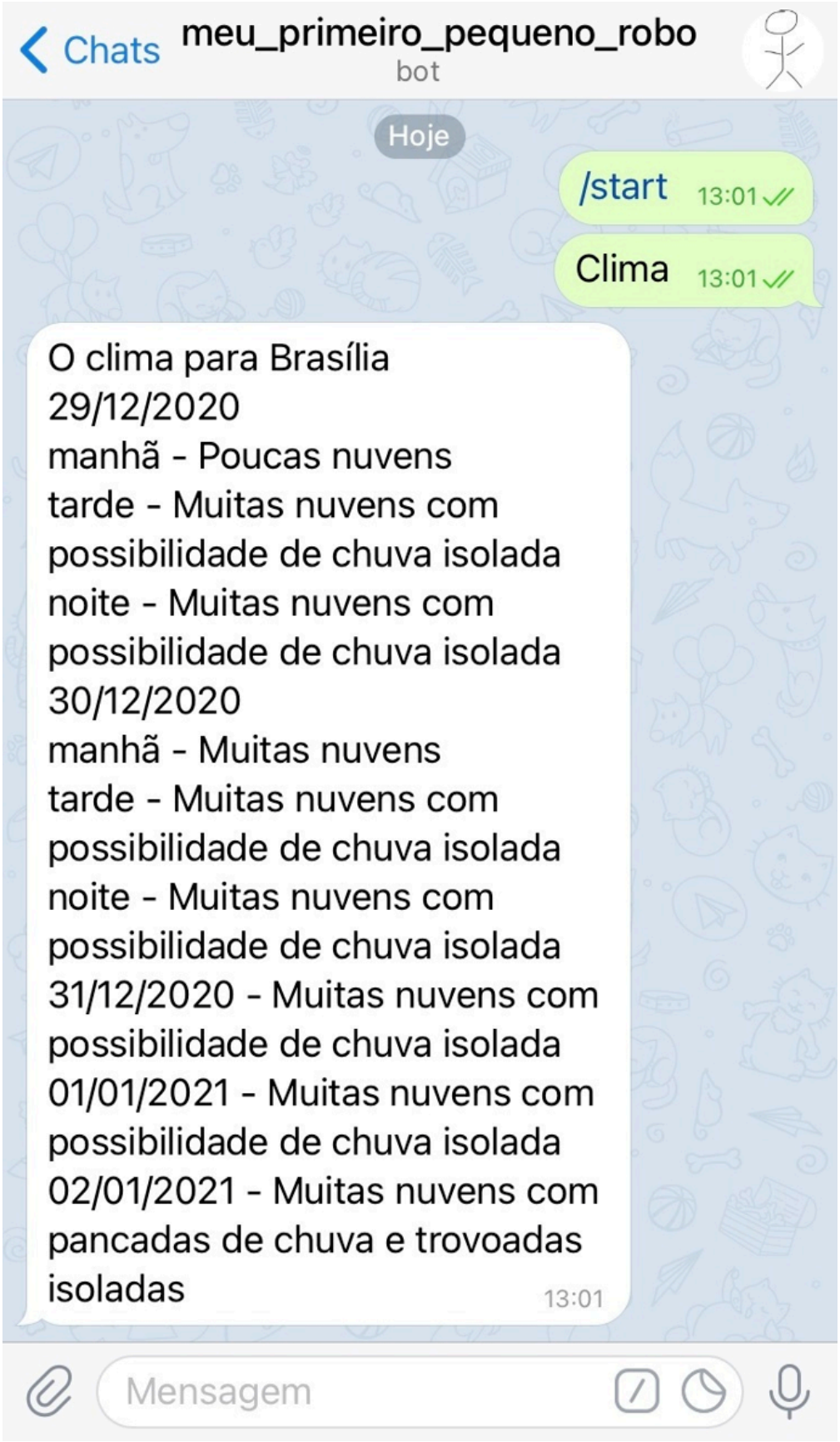

Chats
meu_primeiro_pequeno_robo
bot
Hoje
/start 13:01
Clima 13:01
O clima para Brasília
29/12/2020
manhã - Poucas nuvens
tarde - Muitas nuvens com possibilidade de chuva isolada
noite - Muitas nuvens com possibilidade de chuva isolada
30/12/2020
manhã - Muitas nuvens
tarde - Muitas nuvens com possibilidade de chuva isolada
noite - Muitas nuvens com possibilidade de chuva isolada
31/12/2020 - Muitas nuvens com possibilidade de chuva isolada
01/01/2021 - Muitas nuvens com possibilidade de chuva isolada
02/01/2021 - Muitas nuvens com pancadas de chuva e trovoadas isoladas
13:01
Mensagem

Figura 4.3: A resposta do bot no Telegram ao comando "clima"

Se você analisar bem o que fizemos, ainda há espaço para melhorar muito esse nosso bot!

Uma possível melhoria, por exemplo, é responder a previsão do tempo a partir de uma localização enviada pelo usuário. Para conseguirmos fazer isso, nos próximos capítulos veremos como receber e tratar posições geográficas. Com isso, essa tarefa será fácil e divertida! Siga em frente!

Capítulo 5

Reconhecendo o tipo do arquivo pelo conteúdo

Vamos construir um bot que receberá qualquer tipo de arquivo e retornará o tipo de conteúdo recebido (`content_type`). Vamos verificar essa resposta ao vivo, mesmo que já tenhamos visto no capítulo *O primeiro bot (Olá, Mundo!)* quais são os tipos de arquivo possíveis. Você verificará que, de tantos possíveis, na prática e no dia a dia, para aplicações corriqueiras, usamos poucos deles – mas é sempre bom ter muitas possibilidades, porque isso provê flexibilidade.

Depois, vamos usar esse conhecimento do conteúdo a nosso favor para realizarmos outros processamentos.

Para isso, o bot ficará minúsculo, mas é só o começo:

```
#!/usr/bin/python3
#coding: utf-8
import telepot, time

# telepot.api.set_proxy('http://192.168.0.1:3128',
('usuário','senha'))

def principal(msg):
        content_type, chat_type, chat_id = telepot.glance(msg)

        mensagem = "O tipo de conteúdo recebido foi: "
        mensagem += content_type

        bot.sendMessage(chat_id,mensagem)

bot =
telepot.Bot('1409666004:AAF7eu_YN9Pu8ll0Tx_ey78NyeElJ7jUNMo')
bot.message_loop(principal)
```

```
while 1:
        time.sleep(5)
```

Para testar esse bot, é só executá-lo e mandar qualquer coisa para ele, seja texto, imagens, arquivos etc. Veja os exemplos a seguir:

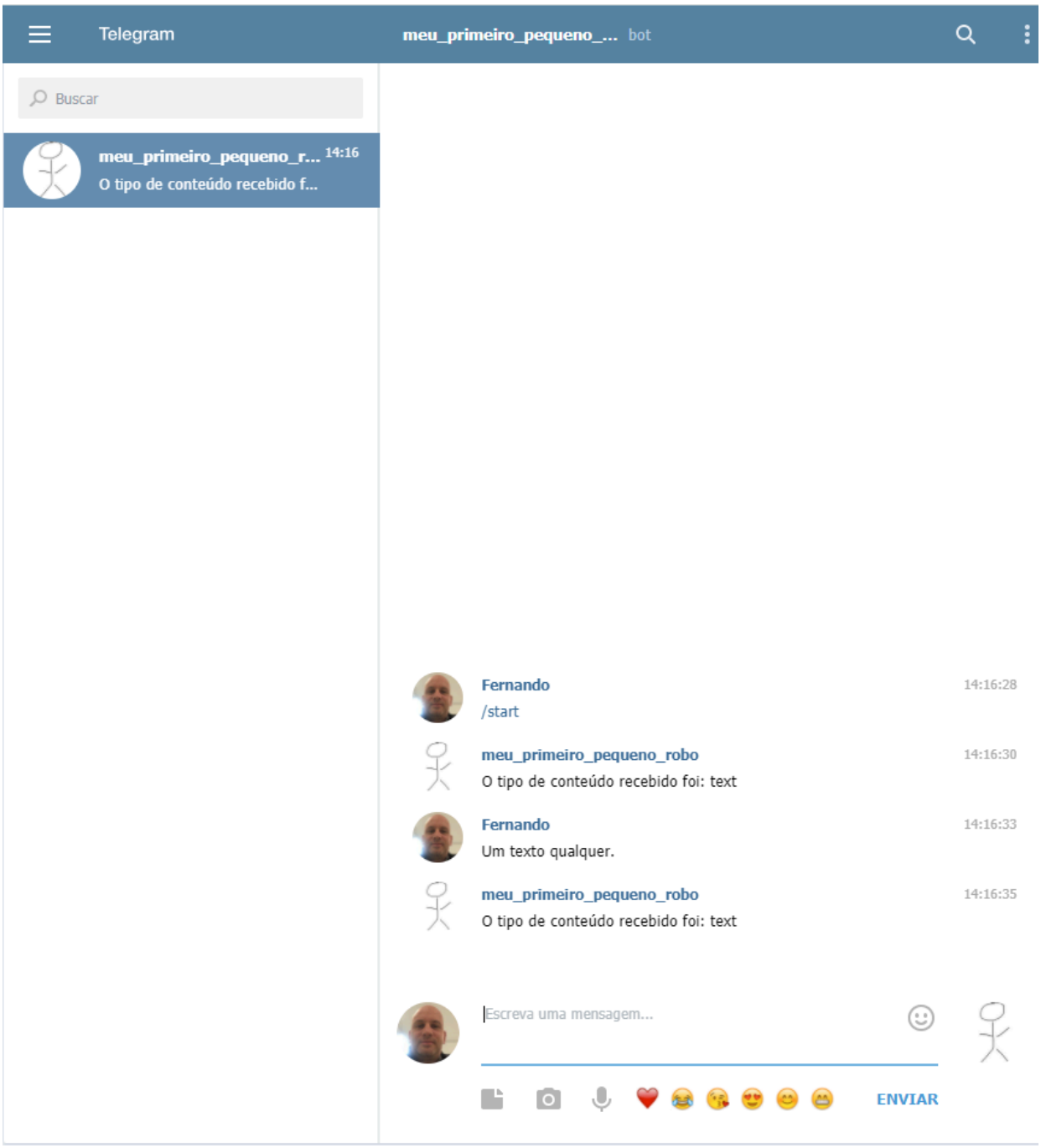


Figura 5.1: Bot recebendo texto 'text'

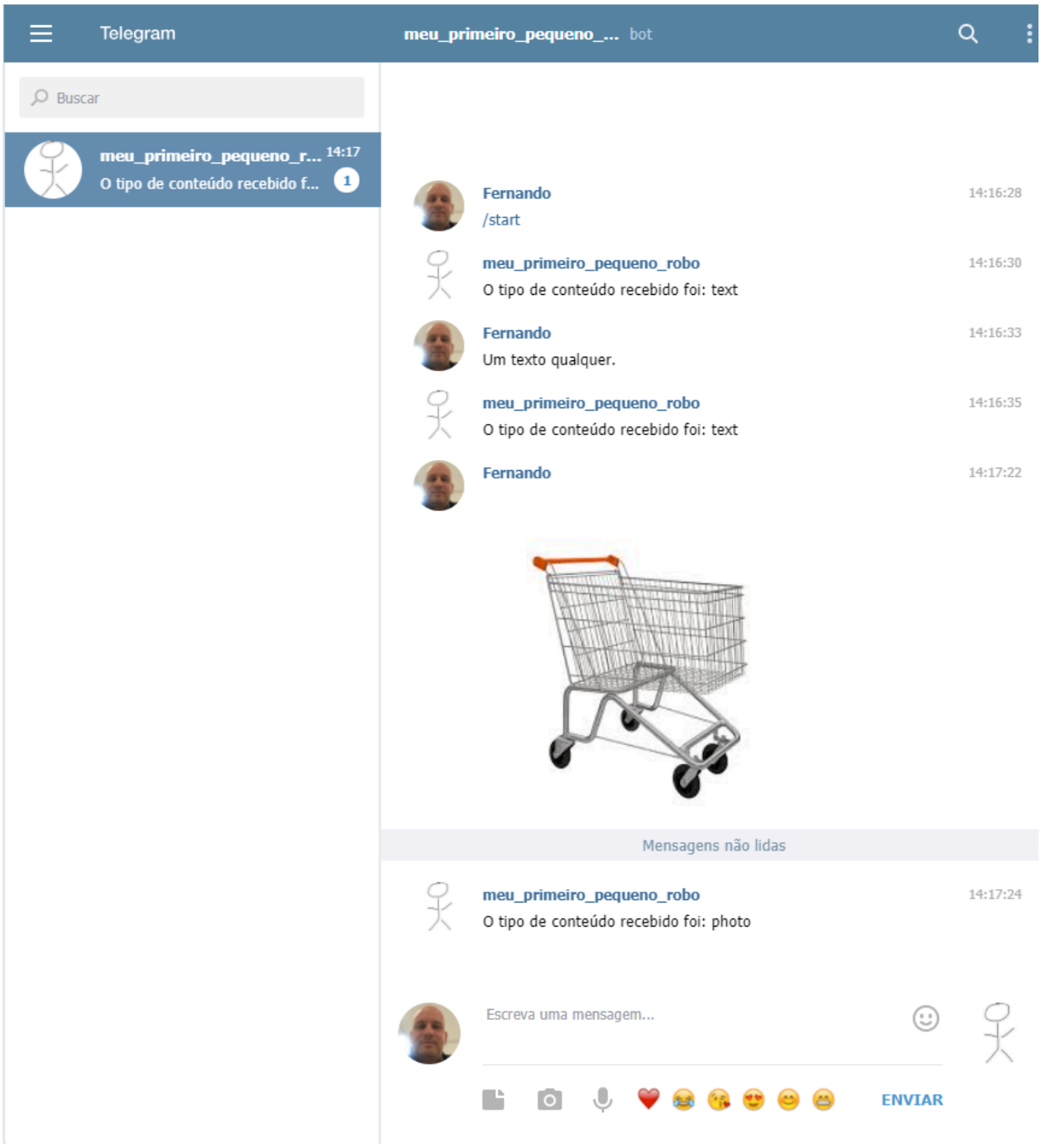


Figura 5.2: Bot recebendo imagem 'photo'

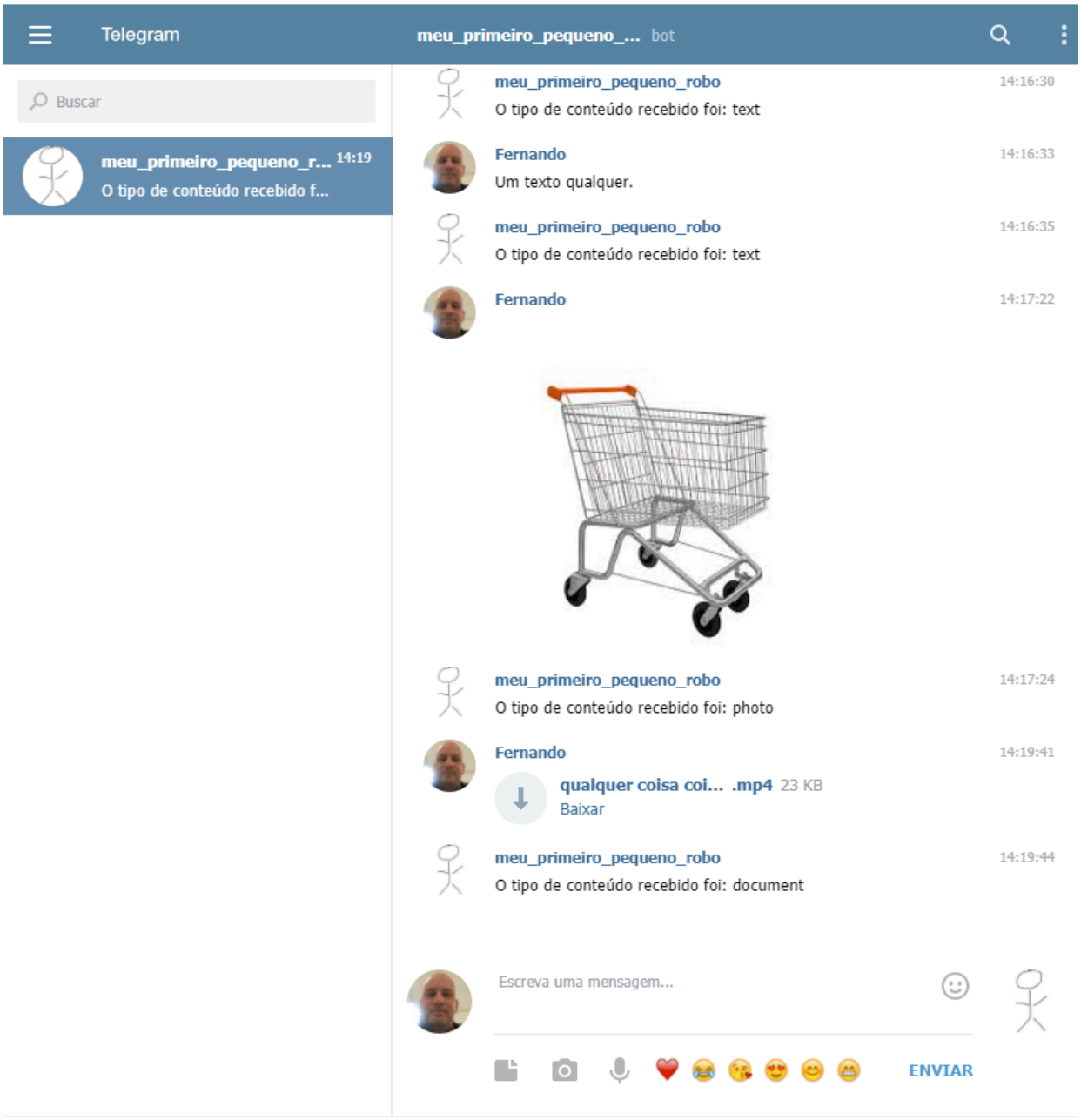


Figura 5.3: Bot recebendo uma mensagem de voz 'voice'

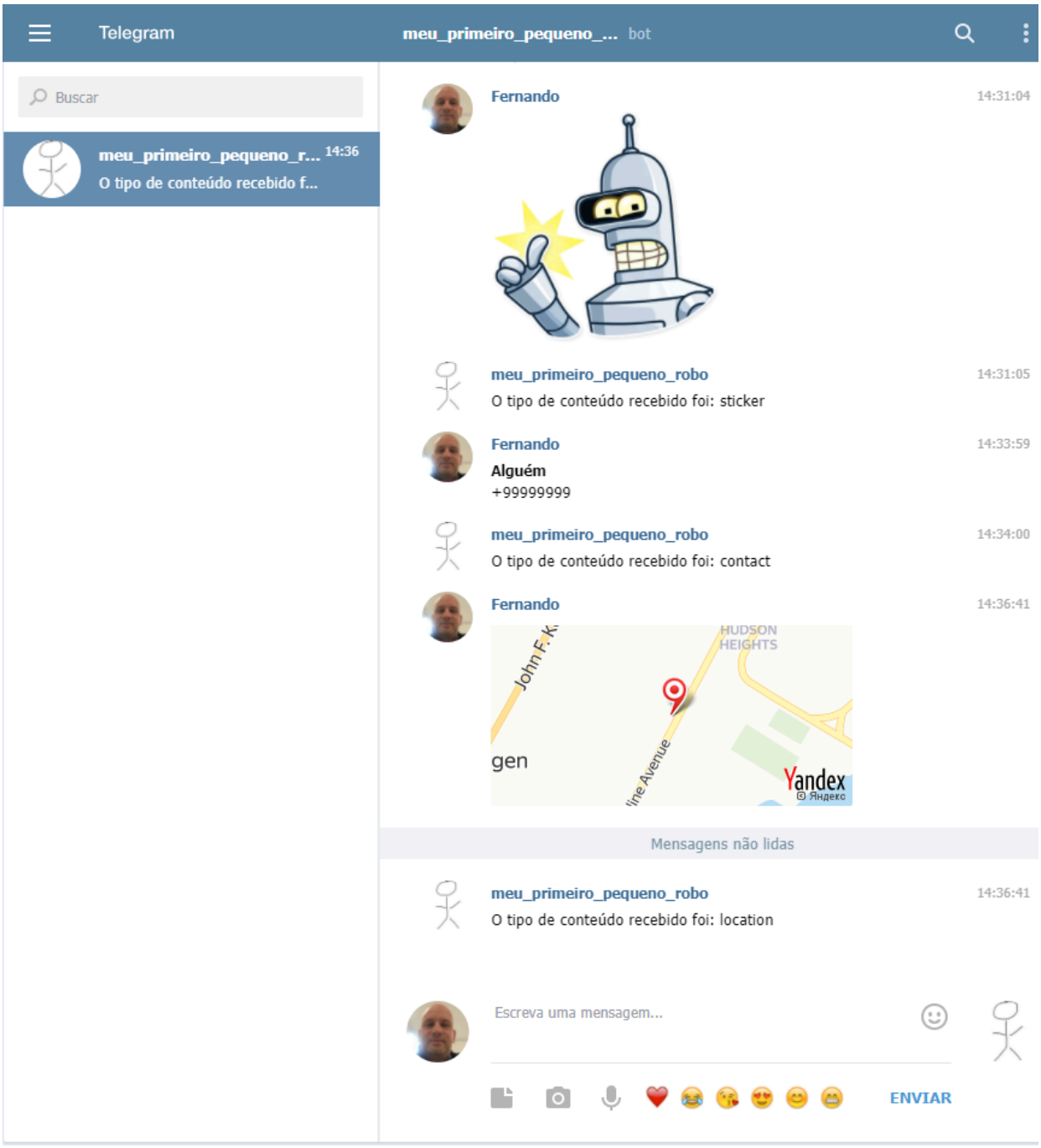


Figura 5.4: Bot recebendo uma localização geográfica 'location'

Não vamos ficar só nisso, claro!

Vamos tentar enviar arquivos no formato do Microsoft Word e Excel para testar:

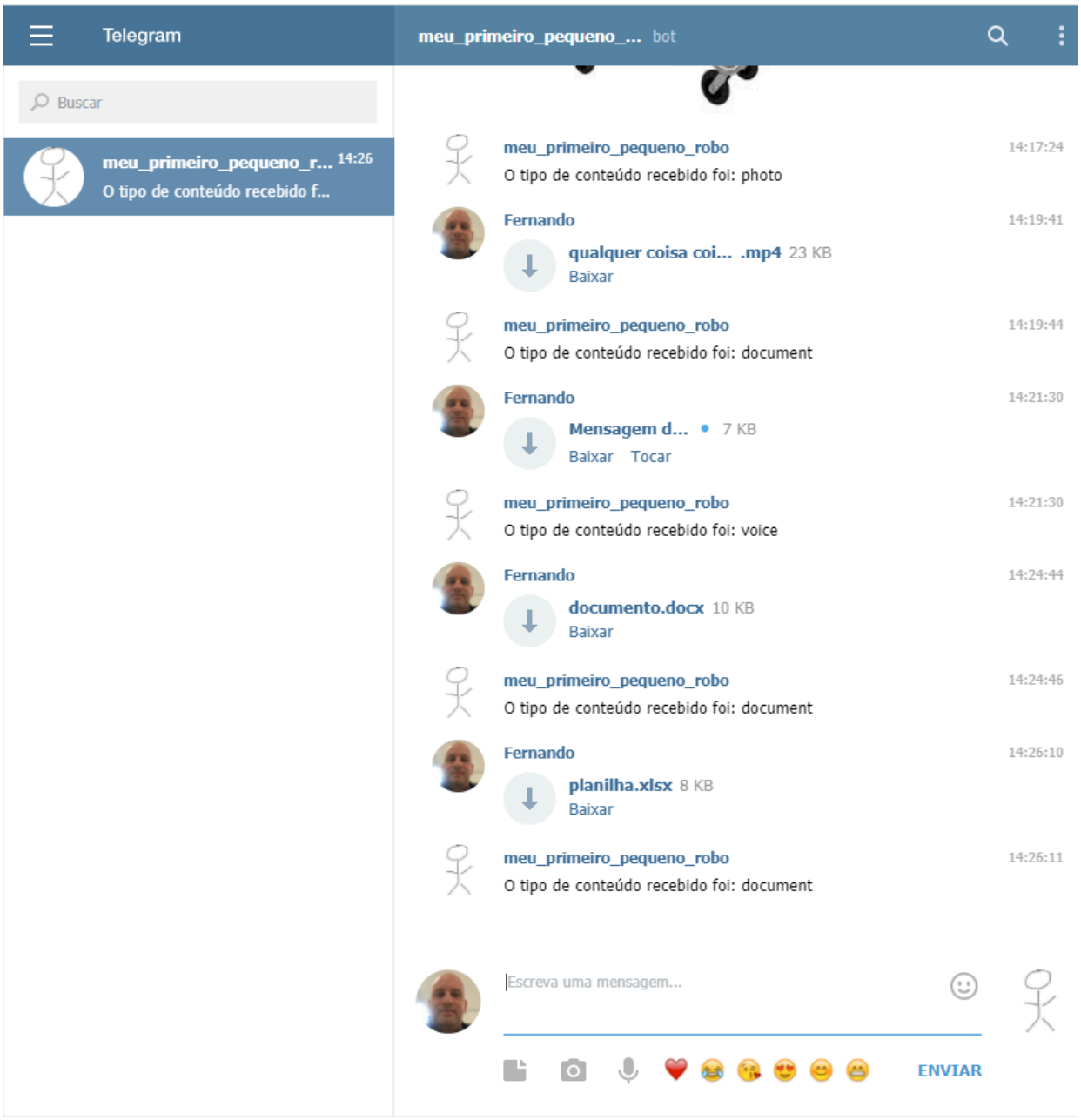


Figura 5.5: Bot recebendo arquivos do tipo 'document'

Veja que, *a priori*, quando recebe arquivos, o bot mostra no `content_type` o tipo `document`, mas não diferencia se é áudio, imagem, documento do Word ou planilha do Excel.

Eis um ponto de melhoria desse bot! Mãos à obra!

O começo você já conhece:

```
#!/usr/bin/python3
#coding: utf-8
import telepot, time, magic

# telepot.api.set_proxy('http://192.168.0.1:3128',
('usuário','senha'))
```

A diferença aqui é a biblioteca **magic**. Essa biblioteca faz mágica! Brincadeiras à parte, essa biblioteca provê classes e métodos para identificação de arquivos pelo seu conteúdo. Isso é possível por meio da verificação dos cabeçalhos dos arquivos.

Para isso, basta fazer a instalação usando `pip install python-magic python-magic-bin libmagic` no Windows ou `pip3 install python-magic python-magic-bin libmagic` no Linux. Se você tem proxy na sua rede, precisa especificá-lo — se for preciso, volte ao capítulo *O que é necessário para continuar* para rever como fazer isso e instalar bibliotecas para o Python.

Com base nisso, nosso código precisará apenas de poucas alterações - na verdade, inclusões - para responder o tipo do arquivo. A função principal para processar as mensagens ficará assim:

```
def principal(msg):
        content_type, chat_type, chat_id = telepot.glance(msg)

        if content_type == 'document':
                bot.download_file(msg[content_type]
['file_id'],msg[content_type]['file_id'])
                mensagem = magic.from_file(msg[content_type]
['file_id'])
                bot.sendMessage(chat_id,mensagem)
```

Com `if content_type == 'document':`, verificamos se o que foi recebido é reconhecido como um `document` pelo bot; caso seja, usamos a linha `bot.download_file(msg[content_type]['file_id'],msg[content_type]['file_id'])` para fazer download do arquivo recebido.

O parâmetro `msg` recebido pela função será um vetor, que trará consigo várias informações que podem ser acessadas, conforme a necessidade.

Com `msg[content_type]['file_id']`, conseguimos obter o nome do arquivo dado pelo Telegram a partir do `content_type` recebido, que, no caso, é verificado se é `document`.

Com o método `download_file` do objeto `bot`, passamos como primeiro parâmetro a identificação do arquivo que queremos baixar, que será o que acabou de ser recebido com `msg[content_type]['file_id']`; com o segundo parâmetro, informamos o destino onde desejamos guardar o arquivo. Como passamos o mesmo parâmetro, ele salvará o arquivo com o mesmo nome que foi recebido. Como não designamos extensão para o arquivo, ele será salvo sem extensão. Por último, como não designamos caminho para a gravação, ele será armazenado na mesma pasta onde está o código-fonte do bot.

Conseguimos o tipo do arquivo com a linha `mensagem = magic.from_file(msg[content_type]['file_id'])`. Basta passar o nome do arquivo (caminho e extensão, se for o caso) para o método `from_file` da classe `magic`, que ele retornará o formato do arquivo que acabou de analisar.

Dessa forma, o código-fonte final do bot ficará assim:

```
#!/usr/bin/python3
#coding: utf-8
import telepot, time, magic

# telepot.api.set_proxy('http://192.168.0.1:3128',
('usuario','senha'))

def principal(msg):
        content_type, chat_type, chat_id = telepot.glance(msg)

        if content_type == 'document':
                bot.download_file(msg[content_type]
['file_id'],msg[content_type]['file_id'])
                mensagem = magic.from_file(msg[content_type]
```

```
['file_id'])
                bot.sendMessage(chat_id,mensagem)

bot =
telepot.Bot('1409666004:AAF7eu_YN9Pu8ll0Tx_ey78NyeElJ7jUNMo')
bot.message_loop(principal)

while 1:
        time.sleep(5)
```

Para testar, vamos reenviar alguns arquivos que foram reconhecidos como `document` anteriormente:

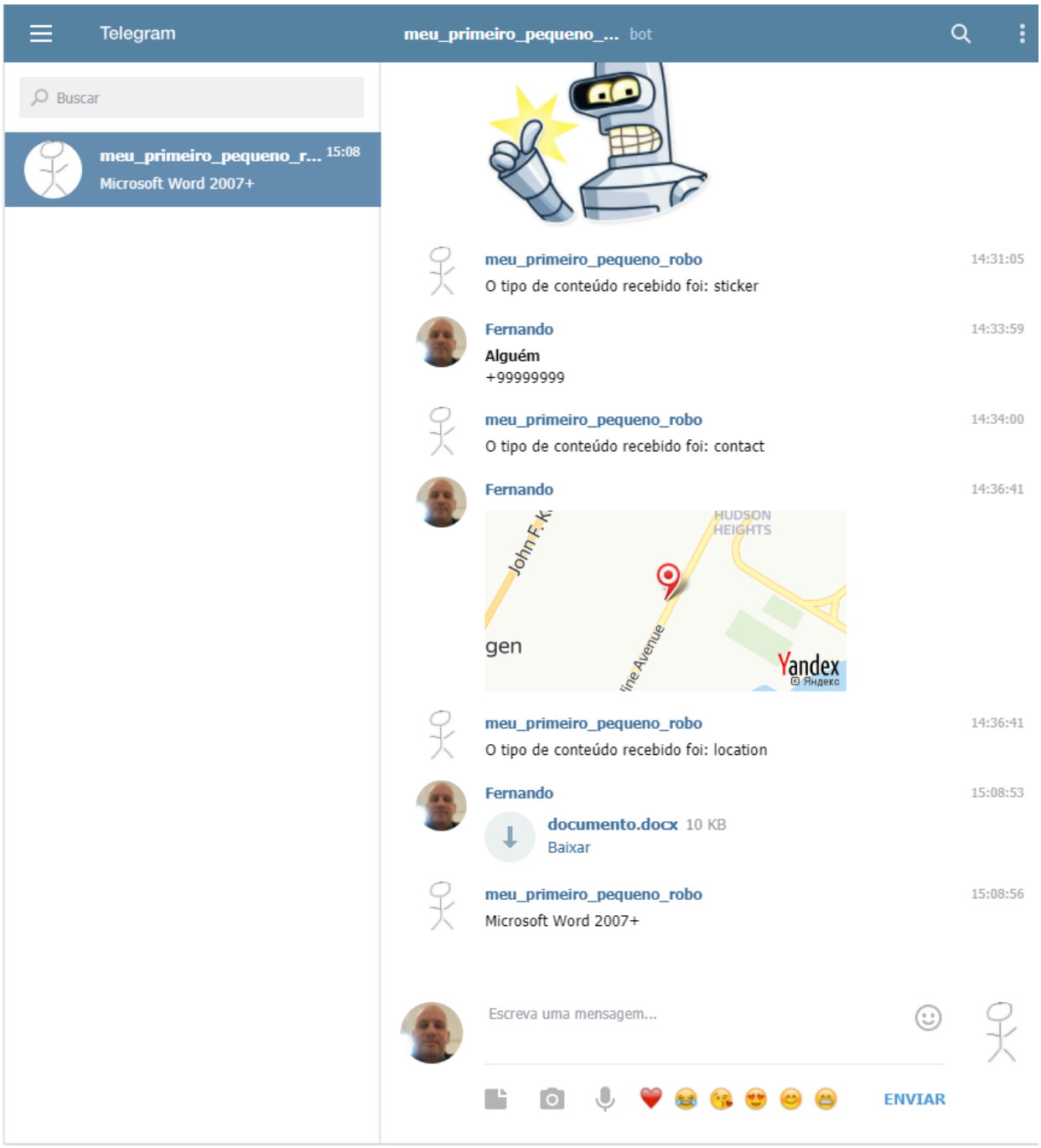


Figura 5.6: Bot identificando o arquivo do tipo Microsoft Word

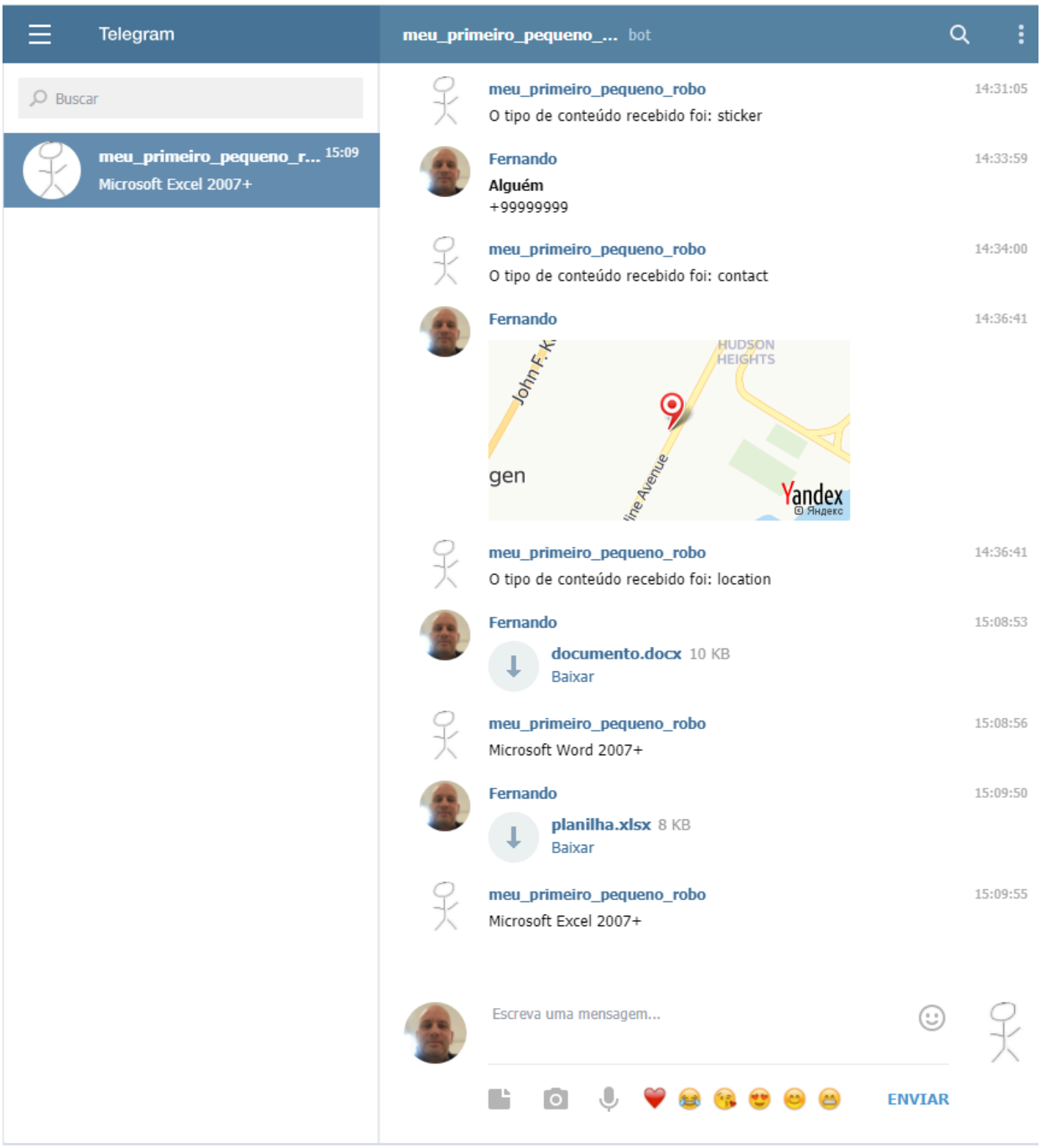


Figura 5.7: Bot identificando o arquivo do tipo Microsoft Excel

Pronto — ou quase pronto!

A partir de agora temos um problema: quando o usuário mandar um arquivo, o bot vai gravá-lo no mesmo local de onde o programa está sendo executado, ou seja, ele está acumulando arquivos em algum

disco, pen drive etc. Isso pode ser indesejável, já que ocupa espaço de armazenamento.

Para resolver esse problema, o bot pode apagar o arquivo depois de analisá-lo. Para isso, vamos incluir uma nova importação e apenas mais uma linha de código. A importação é a biblioteca `os`, que provê facilidades para executar tarefas do sistema operacional (OS, na sigla em inglês), independente de qual seja. A linha `os.remove(msg[content_type]['file_id'])` deverá ser inserida logo depois da do envio da mensagem.

O método remove da classe `os` e realiza a exclusão do arquivo passado como parâmetro.

Além de não desperdiçar espaço de armazenamento, isso garante a privacidade do usuário — fique de olho na LGPD, a **Lei Geral de Proteção de Dados** (Lei Federal n. 13.709, de 14 de agosto de 2018).

Com isso, nosso novo código-fonte completo fica assim:

```
#!/usr/bin/python3
#coding: utf-8
import telepot, time, magic, os

# telepot.api.set_proxy('http://192.168.0.1:3128',
('usuario','senha'))

def principal(msg):
        content_type, chat_type, chat_id = telepot.glance(msg)

        if content_type == 'document':
                bot.download_file(msg[content_type]
['file_id'],msg[content_type]['file_id'])
                mensagem = magic.from_file(msg[content_type]
['file_id'])
                bot.sendMessage(chat_id,mensagem)
                os.remove(msg[content_type]['file_id'])

bot =
telepot.Bot('1409666004:AAF7eu_YN9Pu8ll0Tx_ey78NyeElJ7jUNMo')
bot.message_loop(principal)
```

```
while 1:
        time.sleep(5)
```

E aí? Acha que dá para melhorar ainda mais?

Claro que dá! Podíamos, por exemplo, formatar a mensagem de retorno para o usuário e fazer com que a mensagem "O arquivo que você enviou é do tipo:" seja enviada em negrito e o tipo do arquivo, em texto normal.

Isso é possível ao colocar no método `sendMessage` o tipo de formatação que desejamos:

```
bot.sendMessage(chat_id,[texto],
parse_mode='[estilo_de_formatação]')
```

Em `estilo_de_formatação`, podemos especificar que tipo de formatação será utilizado – MarkdownV2, Markdown ou HTML. Para conhecer a formatação MarkdownV2, um bom local de referência é o Markdown Guide, disponível em http://www.markdownguide.org.

Para formatar o texto, o bloco de código da função principal ficará assim:

```
content_type, chat_type, chat_id = telepot.glance(msg)

        if content_type == 'document':
                bot.download_file(msg[content_type]
['file_id'],msg[content_type]['file_id'])
                mensagem = "*O arquivo que você enviou é do
tipo:* "
                mensagem += magic.from_file(msg[content_type]
['file_id'])

bot.sendMessage(chat_id,mensagem,parse_mode='Markdown')
                os.remove(msg[content_type]['file_id'])
```

Incluímos a linha `mensagem = "**O arquivo que você enviou é do tipo:** "` para mostrar em negrito a mensagem "O arquivo que você

enviou é do tipo:” – em *Markdown*, os asteriscos delimitam o texto a ser mostrado nesse estilo.

Mudamos a linha `mensagem += magic.from_file(msg[content_type]['file_id'])` de `mensagem = magic...` para que o resultado da análise do arquivo pela classe `magic` seja adicionado no final da mensagem “O arquivo que você enviou é do tipo:”.

Ao final da seguinte linha:
`bot.sendMessage(chat_id,mensagem,parse_mode='Markdown'),`
incluímos mais um parâmetro, o `parse_mode`, para indicarmos qual o tipo de formatação que será utilizado. Você também pode testar usando o HTML e o MarkdownV2.

Ao rodar o nosso bot e enviar um arquivo, o resultado será parecido com a imagem a seguir:

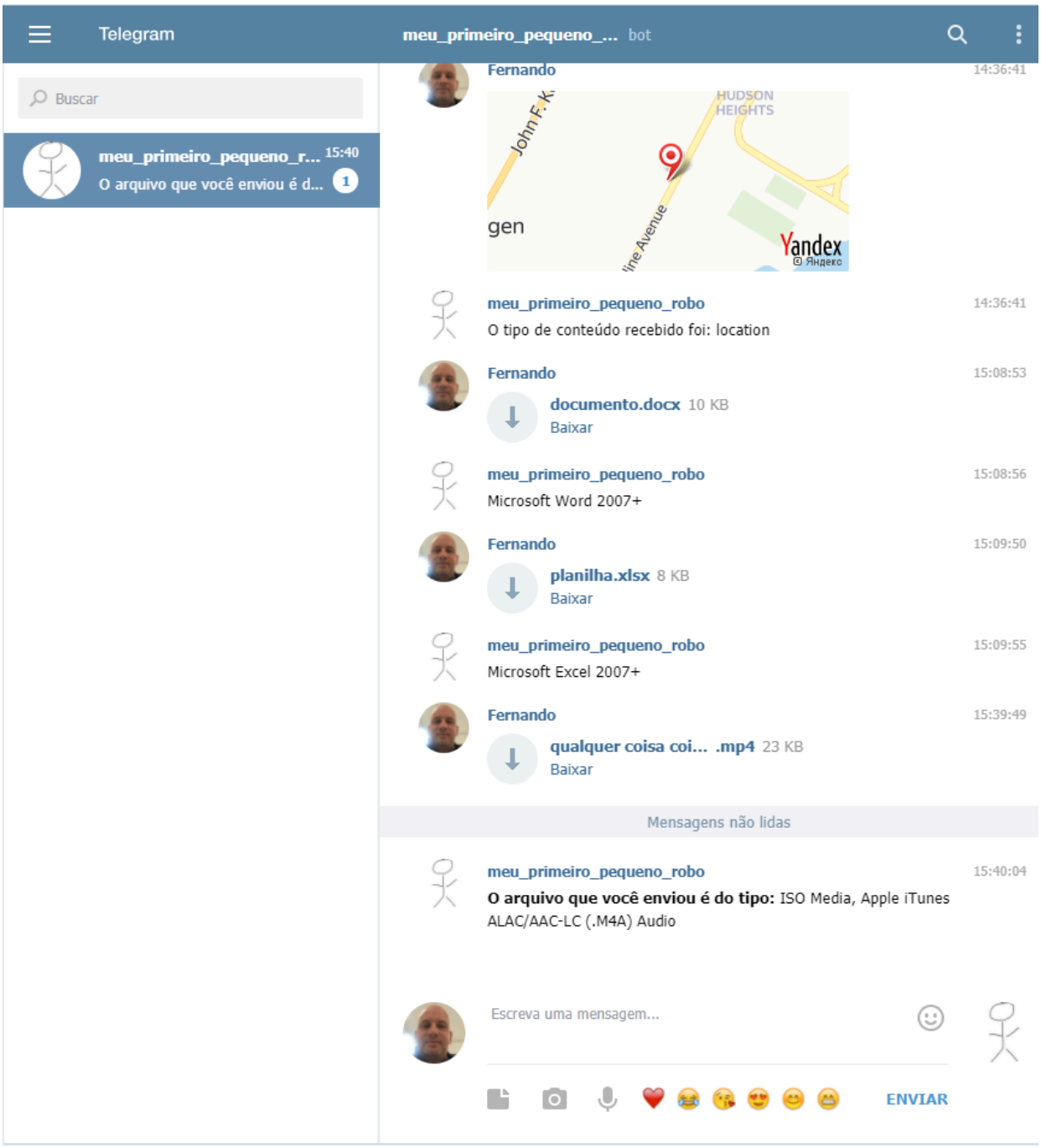


Figura 5.8: Resposta do bot com o texto formatado

Agora é hora de se divertir um pouco com esse bot e, claro, incrementá-lo! Com todo esse conhecimento que temos agora de como reconhecer o formato do arquivo, podemos realizar processamentos mais complexos em cima desses arquivos recebidos

(*lembre-se da LGPD*). Saber que existe uma biblioteca só para nos auxiliar com as tarefas do sistema operacional nos abre um horizonte enorme de possibilidades!

Você tem ideias? Programe-as!

Se até aqui já fizemos tudo isso, siga em frente para ver o que mais podemos fazer!

Capítulo 6

Imagens e reconhecimento de caracteres

6.1 Recebendo e enviando imagens

A ideia deste chatbot é brincar um pouco com imagens: vamos tanto receber quanto enviar imagens.

Primeiro, vamos receber um texto e, a partir dele, gerar uma imagem que será enviada de volta para o usuário. Para isso, precisaremos instalar a biblioteca **Pillow** para o Python.

Depois, vamos fazer um chatbot que recebe uma imagem e retorna todos os caracteres reconhecíveis nela para o usuário. Por exemplo, se você fotografar uma placa de trânsito ou até mesmo um papel com alguma coisa escrita, o bot retornará em formato de texto o que tiver reconhecido como caractere. Nesse segundo bot, vamos precisar da biblioteca **Google Tesseract** para realizar esse processamento.

Agora que você já sabe quais são as ideias, mãos à obra!

Começaremos com a função de receber um texto e retorná-lo como imagem. O primeiro passo é instalar a biblioteca Pillow.

No Windows:

```
pip install Pillow
```

No Linux:

```
pip3 install Pillow
```

Caso você precise de configurações para proxy e não se lembre como faz, dê uma nova olhada no capítulo *O que é necessário para continuar*.

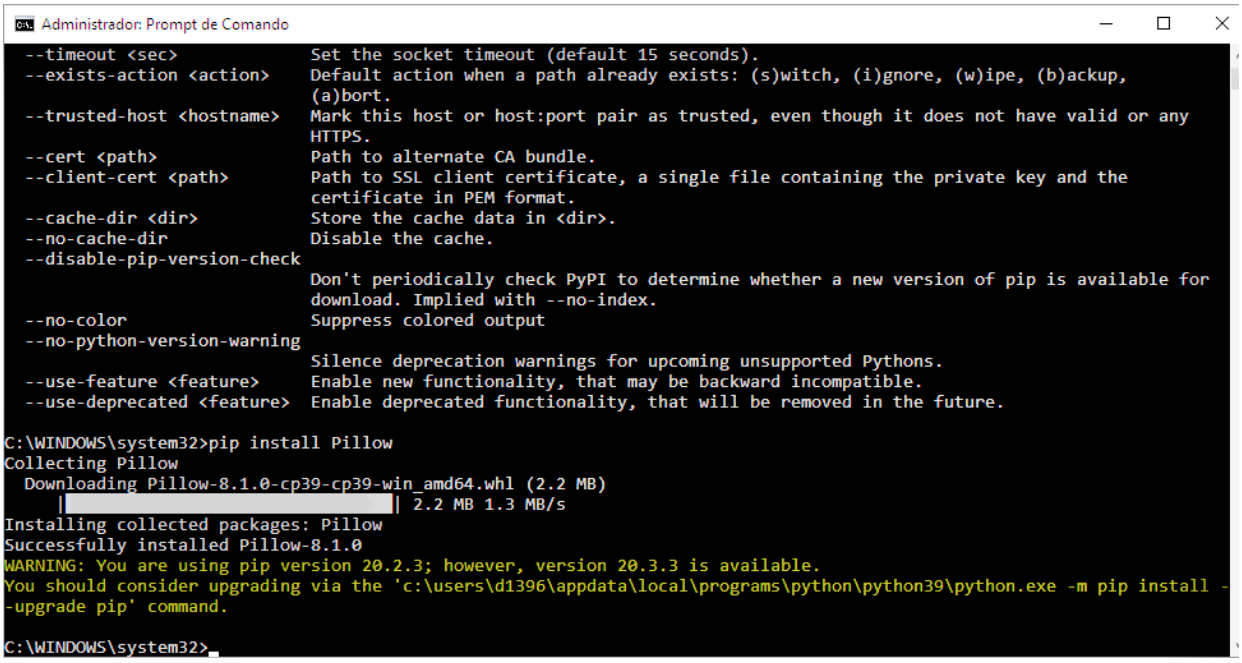


Figura 6.1: Biblioteca Pillow instalada no Windows

Com a biblioteca Pillow instalada, vamos começar a construir nosso bot. Vamos usar o mesmo token de acesso que usamos até agora.

Vamos construí-lo de forma que, quando receber o comando `IMAGEM`, ele responda com uma imagem gerada pelo código programado. O início será assim:

```
#!/usr/bin/python3
#coding: utf-8
import telepot, time
from PIL import Image, ImageDraw

# telepot.api.set_proxy('http://192.168.0.1:3128',
('usuário','senha'))
```

A única coisa diferente do que já fizemos até agora é importar as classes `Image` e `ImageDraw` da biblioteca `PIL` (Pillow) com a linha `from PIL import Image, ImageDraw`. Essas duas classes vão nos fornecer os

métodos que precisaremos para que o bot responda o texto digitado em uma imagem.

A nossa função principal ficará assim:

```
def principal(msg):
        content_type, chat_type, chat_id = telepot.glance(msg)

        if content_type == 'text':
            chat_id = msg['chat']['id']
            mensagem = msg['text']

            if mensagem.upper() == 'IMAGEM':
                img = Image.new('RGB', (100, 30), color = 'red')
                d = ImageDraw.Draw(img)
                img.save('teste.png')

bot.sendPhoto(chat_id,photo=open("teste.png","rb"))
```

Ela é praticamente o que já fizemos nos capítulos anteriores. O que muda é a estrutura `if` para processarmos o comando:

```
if mensagem.upper() == 'IMAGEM':
    img = Image.new('RGB', (100, 30), color = 'red')
    img.save('teste.png')
    bot.sendPhoto(chat_id,photo=open("teste.png","rb"))
```

Com a linha `img = Image.new('RGB', (100, 30), color = 'red')`, caso a mensagem enviada seja `IMAGEM` (lembrando que convertemos para maiúsculas para que o usuário digite como quiser), o bot gerará uma imagem de tamanho 100x30 pixels, na cor vermelha, com a linha `img = Image.new('RGB', (100, 30), color = 'red')`.

Para a cor, podemos usar o nome ou a combinação RGB. Por exemplo, para o vermelho, essa linha também poderia ser assim:

```
img = Image.new('RGB', (100, 30), color = (255,0,0))
```

Sendo que o primeiro número de `color` é o vermelho (R), o segundo, o verde (G), e o terceiro, o azul (B). Com isso, podemos criar qualquer cor possível e desejada.

Com a linha `img.save('teste.png')`, salvamos a imagem criada pelo bot no arquivo teste.png. Como não indicamos o caminho, ela será salva no mesmo diretório de onde o programa está sendo executado. Precisamos criar o arquivo para que possamos enviá-lo de volta ao usuário, com a linha

`bot.sendPhoto(chat_id,photo=open("teste.png","rb"))`.

No parâmetro `photo`, abrimos o arquivo da imagem com permissões de leitura (r) e no formato binário (b). O código completo ficará assim:

```
#!/usr/bin/python3
#coding: utf-8
import telepot, time
from PIL import Image, ImageDraw

# telepot.api.set_proxy('http://192.168.0.7:3128',
('usuário','senha'))

def principal(msg):
        content_type, chat_type, chat_id = telepot.glance(msg)

        if content_type == 'text':
            chat_id = msg['chat']['id']
            mensagem = msg['text']

            if mensagem.upper() == 'IMAGEM':
                img = Image.new('RGB', (100, 30), color = 'red')
                img.save('teste.png')

bot.sendPhoto(chat_id,photo=open("teste.png","rb"))

bot =
telepot.Bot('1409666004:AAF7eu_YN9Pu8ll0Tx_ey78NyeElJ7jUNMo')
bot.message_loop(principal)

while 1:
    time.sleep(5)
```

Executando o programa e enviando para o bot, via Telegram, a mensagem `IMAGEM`, veja o que acontece:

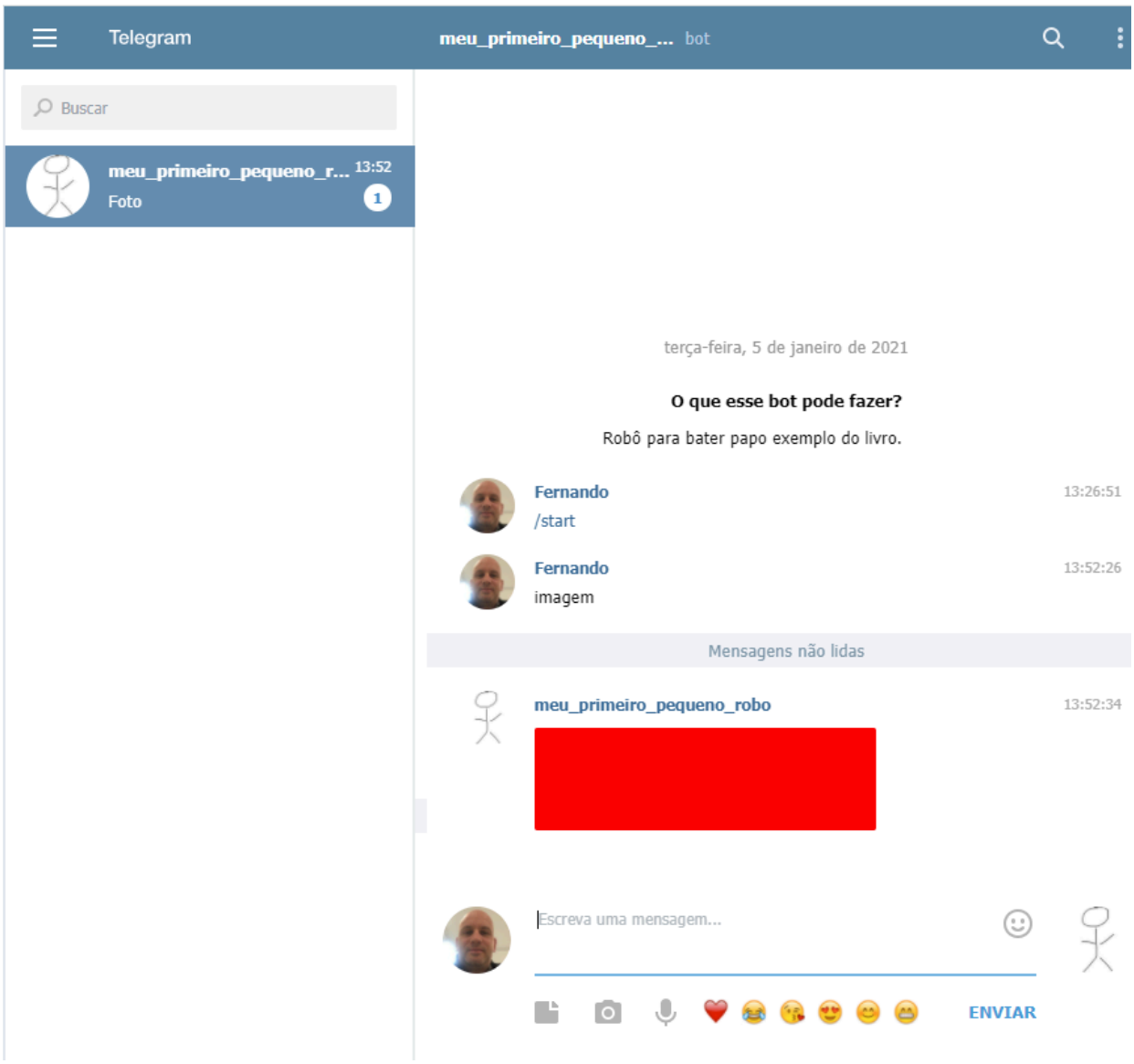


Figura 6.2: Resposta do bot com uma imagem: um retângulo de 100x30 pixels na cor vermelha

Agora, vamos incluir um texto nessa imagem?

Para incluir um texto na imagem, incluímos uma linha onde armazenamos a imagem criada em uma variável, no caso a variável `d`:

```
d = ImageDraw.Draw(img)
```

Depois usamos o método `text` com os parâmetros: posição *x* e *y* de onde o texto deve começar a ser desenhado, o texto a ser exibido e a cor de preenchimento do texto. A linha fica assim:

```
d.text((10,10), "IMAGEM", fill=(255,255,255))
```

Neste caso, estamos preenchendo o nosso texto `IMAGEM` na cor branca e fazendo com que seja iniciado nos pixels `x = 10` e `y = 10` a partir do canto esquerdo superior da imagem.

Dessa forma, o código-fonte completo da função principal, com essas duas inclusões, fica assim:

```
def principal(msg):
        content_type, chat_type, chat_id = telepot.glance(msg)

        if content_type == 'text':
            chat_id = msg['chat']['id']
            mensagem = msg['text']

            if mensagem.upper() == 'IMAGEM':
                img = Image.new('RGB', (100, 30), color = 'red')
                d = ImageDraw.Draw(img)
                d.text((10,10), "IMAGEM", fill=(255,255,255))
                img.save('teste.png')

bot.sendPhoto(chat_id,photo=open("teste.png","rb"))
```

Execute seu programa e, no Telegram, vamos enviar a mensagem `IMAGEM` novamente para ver o que acontece:

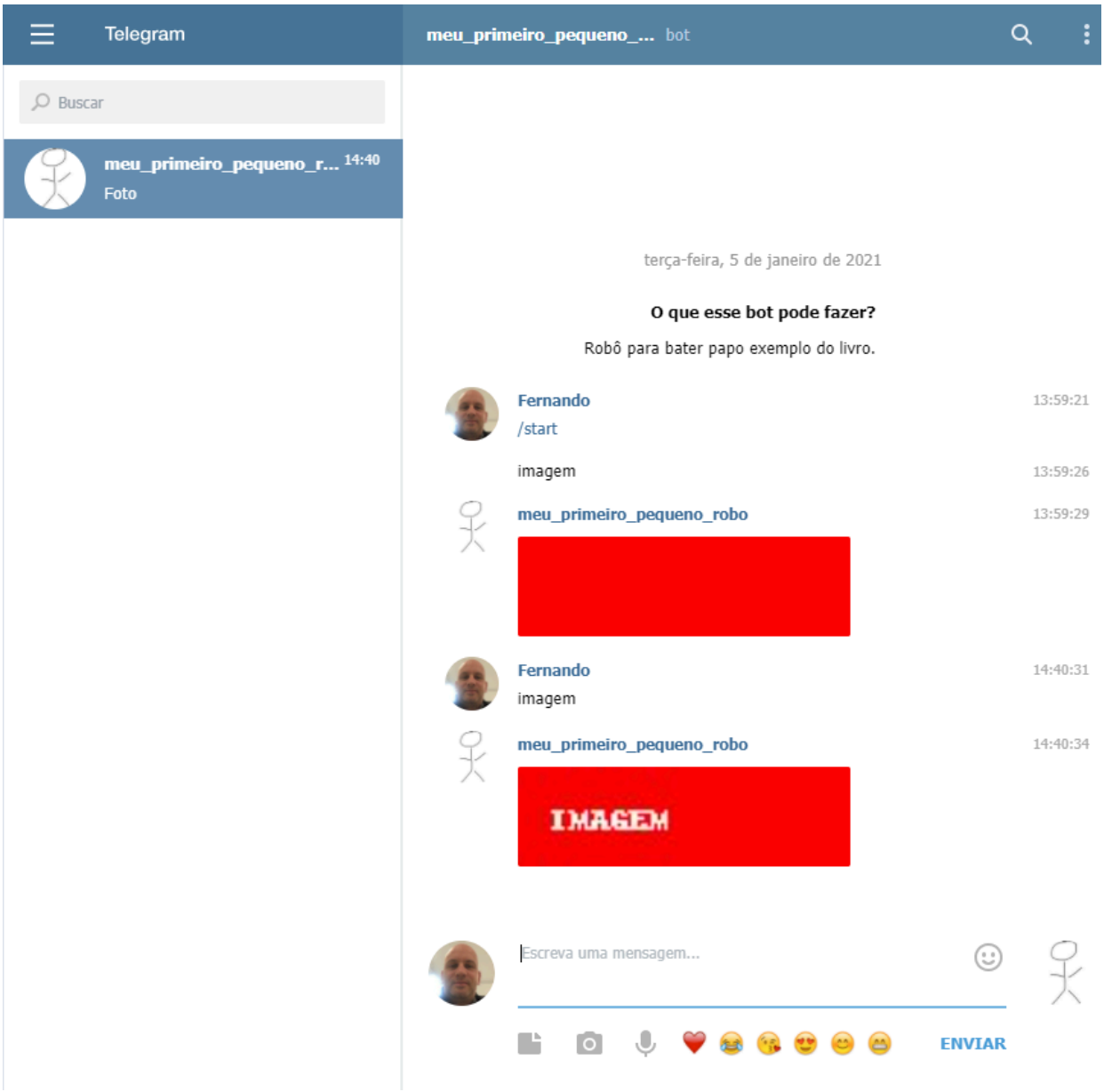


Figura 6.3: A imagem enviada pelo bot: um retângulo vermelho com a palavra IMAGEM escrita em branco

Mas ainda não é o que queremos. A primeira coisa que vamos ajustar é dar ao usuário a opção de escolher a fonte e o tamanho da letra.

Podemos fazer isso importando a classe `ImageFont` da biblioteca PIL (Pillow). A linha de importação dessa biblioteca ficará assim:

```
from PIL import Image, ImageDraw, ImageFont
```

Com isso, escrevemos a seguinte linha para escolhermos a fonte no primeiro parâmetro e, no segundo, o tamanho da letra:

```
fnt = ImageFont.truetype('C:\Windows\Fonts\calibri.ttf', 15)
```

Essa linha fará com que os dados da fonte escolhida sejam armazenados na variável `fnt`. Note que o caminho está informado para sistemas Windows. Para sistemas Linux, por exemplo, o caminho para as fontes é `/usr/share/fonts` e os nomes das fontes podem variar entre os sistemas operacionais, inclusive entre versões diferentes do mesmo sistema operacional. Você terá que verificar a extensão da fonte e o caminho no seu sistema operacional antes de informá-los.

A fonte na qual o texto será efetivamente escrito na imagem deve ser informada na linha do método `text`:

```
d.text((10,10), "IMAGEM", font=fnt, fill=(255,255,255))
```

Indicamos o parâmetro `font=fnt`, onde `font=` é o parâmetro e `fnt`, a variável onde armazenamos os dados da fonte escolhida e o tamanho da letra.

O código-fonte completo, até agora, está assim:

```
#!/usr/bin/python3
#coding: utf-8
import telepot, time
from PIL import Image, ImageDraw, ImageFont

# telepot.api.set_proxy('http://192.168.0.7:3128',
('usuário','senha'))

def principal(msg):
        content_type, chat_type, chat_id = telepot.glance(msg)

        if content_type == 'text':
            chat_id = msg['chat']['id']
            mensagem = msg['text']

            if mensagem.upper() == 'IMAGEM':
                img = Image.new('RGB', (100, 30), color = 'red')
```

```
                fnt =
ImageFont.truetype('C:\Windows\Fonts\calibri.ttf', 15)
                d = ImageDraw.Draw(img)
                d.text((10,10), "IMAGEM", font=fnt, fill=
(255,255,255))
                img.save('teste.png')

bot.sendPhoto(chat_id,photo=open("teste.png","rb"))

bot =
telepot.Bot('1409666004:AAF7eu_YN9Pu8ll0Tx_ey78NyeElJ7jUNMo')
bot.message_loop(principal)

while 1:
    time.sleep(5)
```

Mais uma vez, execute seu programa e envie a mensagem `IMAGEM` para o bot no Telegram. Teremos o seguinte resultado:

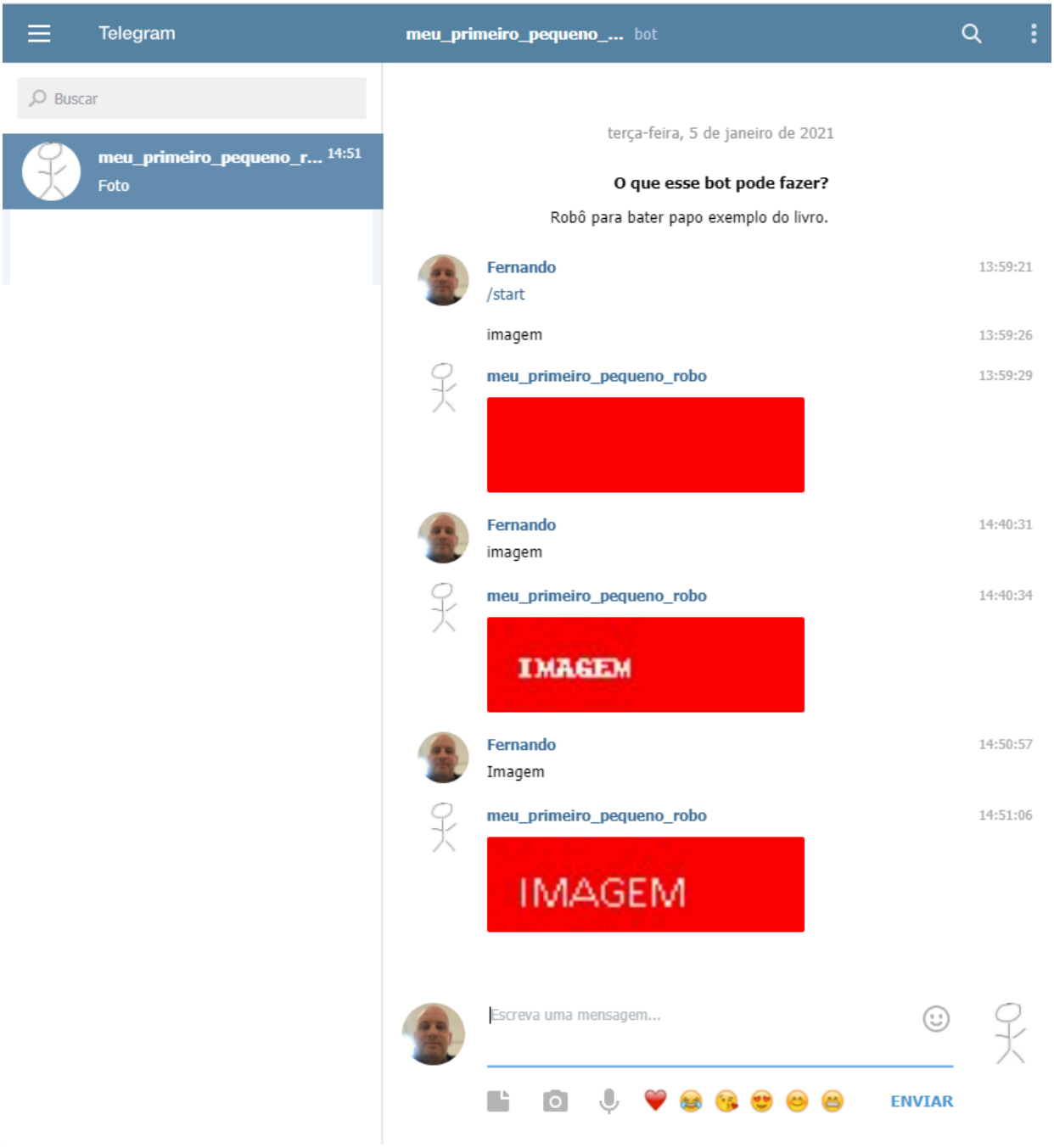


Figura 6.4: Texto com a fonte e tamanho escolhidos

Entretanto, essa ainda não é a ideia! Lembre-se lá do começo do capítulo: nosso objetivo é fazer com que o usuário mande um texto e esse texto seja escrito na imagem enviada como resposta pelo bot.

Por exemplo, ele enviará `IMAGEM TEXTO` e o texto colocado na sequência do comando `IMAGEM` será escrito na imagem e enviado de volta ao usuário.

Faremos isso usando o método `split()` da classe `String` do Python, que retorna a string em um vetor a partir de um separador informado. Mas, caso não seja informado, a separação é feita através do caractere espaço em branco.

O método `split` pode receber dois parâmetros, que são opcionais: o separador, que indica o caractere a ser usado na separação da string em suas componentes; e o máximo de separações, que indica quantas separações deve-se fazer, sendo que o valor -1 indica que deve haver tantas separações quanto possíveis na ocorrência do caractere separador.

Resumindo, o método `split` tem a seguinte sintaxe:
`split([separador],[máx. separações].`

Se mantivermos a linha `if mensagem.upper() == 'IMAGEM':`, essa condição nunca será verdadeira, pois `mensagem.upper()` será igual, conforme o exemplo anterior, `IMAGEM TEXTO` e não somente `IMAGEM`.

Para resolver isso, usamos a técnica de *substring* para "cortar" apenas uma parte da mensagem que nos interessa.

Como `IMAGEM` tem 6 caracteres, podemos selecioná-los apenas usando `mensagem.upper()[0:6]`. Com isso somente serão retornados os caracteres da `String` entre o primeiro (0) e o 5º (6).

Na linha `texto = mensagem.split()`, a variável `texto` será um vetor que conterá todas as palavras separadas por espaço. No caso de enviar, por exemplo, `IMAGEM Fernando`, a palavra `IMAGEM` estará em `texto[0]`, primeira posição do vetor, e `Fernando` em `texto[1]`, segunda posição do vetor.

Sabendo disso, alteramos a linha onde incluímos o texto na imagem para:

```
d.text((10,10), texto[1], font=fnt, fill=(255,255,255))
```

O parâmetro que especifica o texto a ser grafado na imagem ficará como `texto[1]`, ou seja, o que estiver escrito após o primeiro espaço em branco na mensagem.

Dessa forma, nosso código-fonte final ficou assim:

```
#!/usr/bin/python3
#coding: utf-8
import telepot, time
from PIL import Image, ImageDraw, ImageFont

# telepot.api.set_proxy('http://192.168.0.1:3128',
('usuário','senha'))

def principal(msg):
        content_type, chat_type, chat_id = telepot.glance(msg)

        if content_type == 'text':
            chat_id = msg['chat']['id']
            mensagem = msg['text']

            if mensagem.upper()[0:6] == 'IMAGEM':
                texto = mensagem.split()
                img = Image.new('RGB', (100, 30), color = 'red')
                fnt =
ImageFont.truetype('C:\Windows\Fonts\calibri.ttf', 15)
                d = ImageDraw.Draw(img)
                d.text((10,10), texto[1], font=fnt, fill=
(255,255,255))
                img.save('teste.png')

bot.sendPhoto(chat_id,photo=open("teste.png","rb"))

bot =
telepot.Bot('1409666004:AAF7eu_YN9Pu8ll0Tx_ey78NyeElJ7jUNMo')
bot.message_loop(principal)

while 1:
    time.sleep(5)
```

Pronto! Primeira missão cumprida! Agora é só usar a sua imaginação e você fará coisas incríveis! Porém, ainda nos falta fazer o contrário: enviarmos uma imagem ao bot e ele nos dizer se há algo escrito nessa imagem. Topa? Vamos em frente.

6.2 Reconhecimento de caracteres

Antes de colocarmos a mão na massa, precisaremos instalar o **Google Tesseract-OCR**. O Tesseract-OCR é um conjunto de ferramentas para reconhecimento de caracteres mantido pelo Google. Ele já traz incorporadas todas as funcionalidades e ferramentas necessárias para que o reconhecimento de caracteres seja possível.

Para a instalação no Windows, é necessário ir até o endereço https://github.com/UB-Mannheim/tesseract/wiki e baixar a instalação em sua última versão.

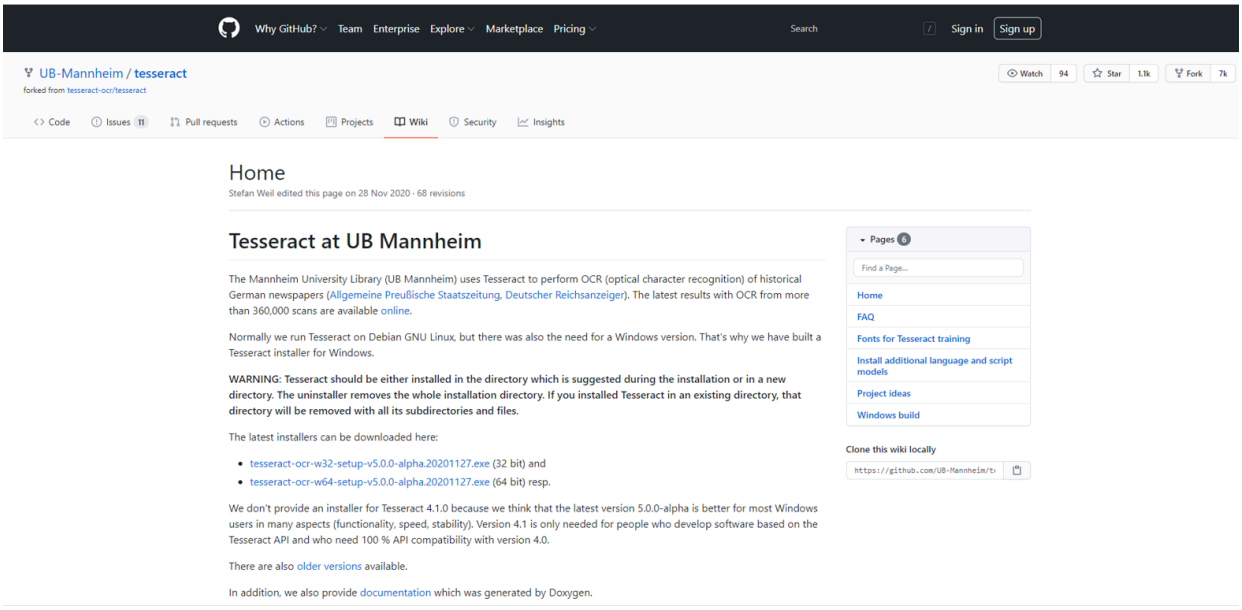


Figura 6.5: O site para download da instalação do Tesseract para Windows

Faça download da instalação específica para a arquitetura de seu computador, 32 ou 64-bit. Depois do download realizado, execute a

instalação. A instalação do Google Tesseract-OCR não está disponível em português.

Durante a instalação, os termos da licença para uso serão apresentados. Sempre salientamos que é importante ler com atenção os termos e condições para uso, reprodução e distribuição antes de simplesmente aceitá-los, pois isso pode evitar problemas futuros com direito de propriedade intelectual.

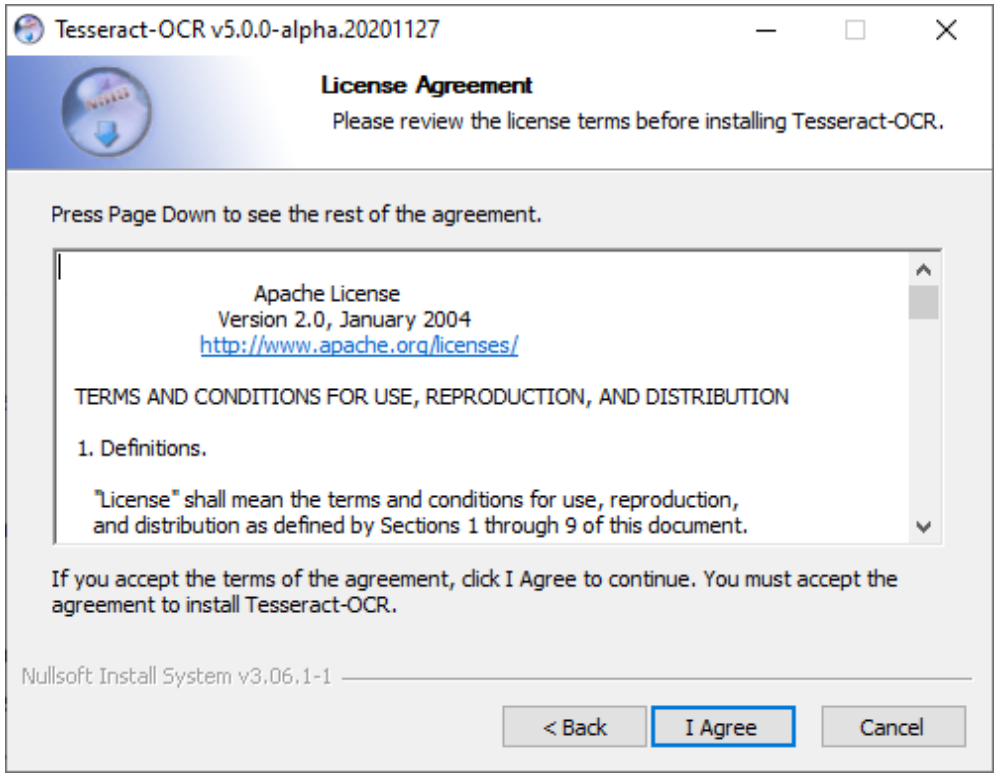


Figura 6.6: A tela de aceite dos termos de licença

Haverá a opção de instalar o Tesseract apenas para o seu usuário no computador ou para todos os usuários. Selecione a opção que melhor lhe convir e continue.

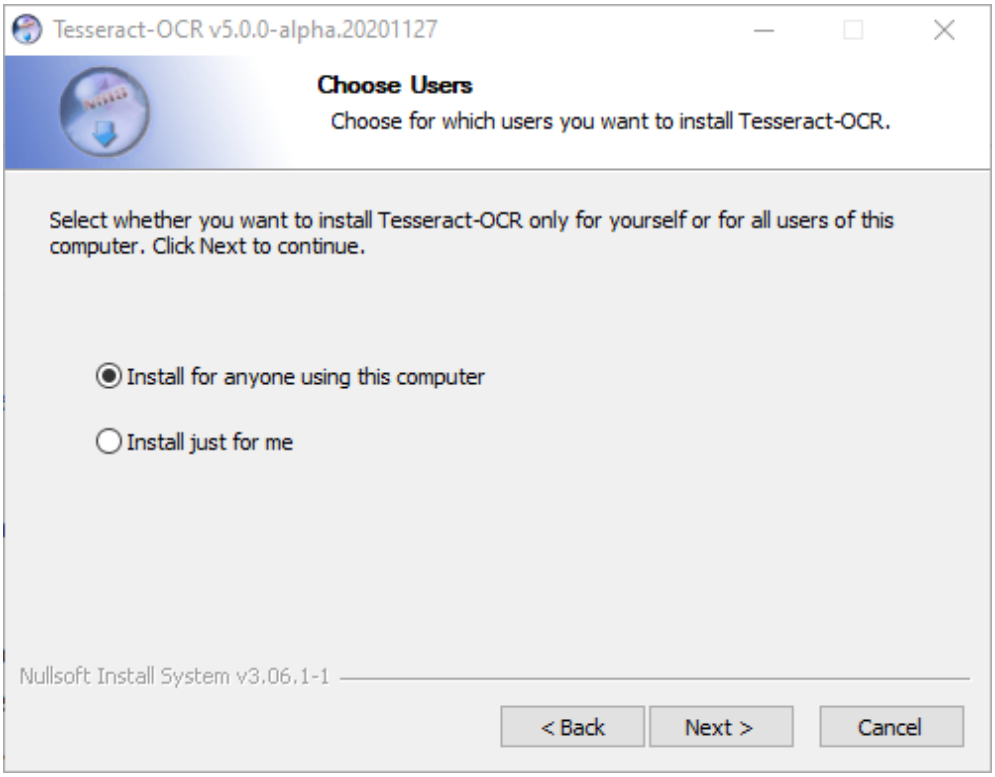


Figura 6.7: Seleção para instalar para todos os usuários ou apenas para o seu

Você pode escolher os componentes que deseja instalar – aqui, precisamos ter atenção ao que vamos selecionar. A própria instalação já sugere a inclusão dos componentes *ScrollView* (visualização), *Training Tools* (ferramentas de treinamento), *Shortcuts creation* (criação de atalhos) e *Language data* (dados de linguagem), mas não traz nada selecionado para *Additional script data* (dados de escrita adicionais) e *Additional language data* (dados adicionais de linguagem).

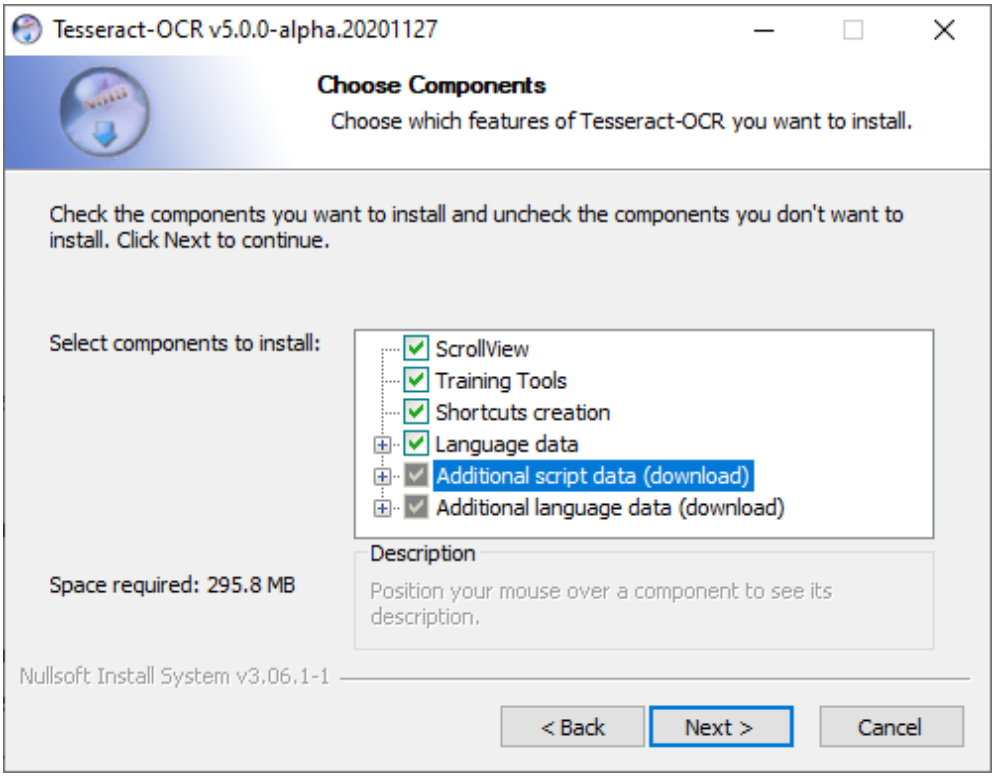


Figura 6.8: A janela para seleção de componentes a serem instalados

Para que o Tesseract consiga processar palavras em português, selecione, em *Additional script data*, a opção *Latin script* (escrita latina); e, em *Additional language data*, selecione a opção *Portuguese* (português).

Feito tudo isso, é só continuar até chegar o momento de escolher o caminho para a instalação, o qual sugerimos deixar no padrão. Mas atenção: é importante anotar o local onde você está instalando o Tesseract-OCR, pois vamos usar essa informação posteriormente em nosso programa.

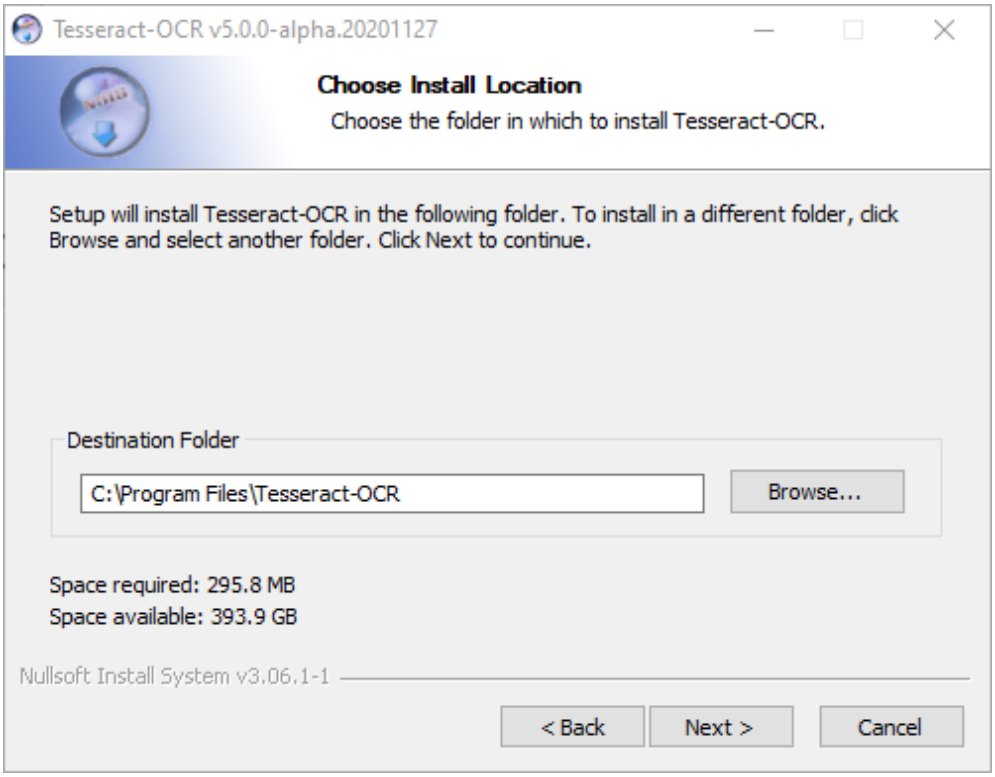


Figura 6.9: Janela para seleção do local da instalação

Finalmente, selecione onde serão criados os atalhos (se desejá-los) e prossiga com a instalação até a conclusão.

Pronto! O Tesseract-OCR está finalmente instalado!

Só mais um passo e teremos tudo o que precisamos à mão: vamos instalar a biblioteca **Pytesseract** pela linha de comandos.

A Pytesseract é uma biblioteca para reconhecimento ótico de caracteres (OCR, na sigla em inglês). Ela faz a interface entre o Python e o Tesseract-OCR. Para realizar a instalação, basta executar os comandos adequados para o seu sistema operacional.

Para o Windows:

```
pip install pytesseract
```

Para o Linux:

```
apt-get install libleptonica-dev tesseract-ocr tesseract-ocr-dev
libtesseract-dev
pip3 install pytesseract
```

Com tudo preparado, vamos ao nosso programa.

```
#!/usr/bin/python3
#coding: utf-8
import telepot, time
from PIL import Image
import pytesseract

# telepot.api.set_proxy('http://192.168.0.1:3128',
('usuário','senha'))

def principal(msg):
        content_type, chat_type, chat_id = telepot.glance(msg)

        if content_type == 'photo':
                caminho = ""
                imagem = msg[content_type][-1]['file_unique_id']
+ ".jpg"
                bot.download_file(msg[content_type][-1]
['file_id'], caminho + imagem)
                pytesseract.pytesseract.tesseract_cmd =
'C:\\Program Files\\Tesseract-OCR\\tesseract.exe'
                tessdata_dir_config = '--tessdata-dir
"C:\\Program Files\\Tesseract-OCR\\tessdata"'
                foto = Image.open(imagem + caminho)
                texto = pytesseract.image_to_string(foto,
lang='por', config=tessdata_dir_config)
                bot.sendMessage(chat_id,"O texto da imagem:\n" +
texto)

bot =
telepot.Bot('1409666004:AAF7eu_YN9Pu8ll0Tx_ey78NyeElJ7jUNMo')
bot.message_loop(principal)

while 1:
        time.sleep(5)
```

Como você pode verificar, o começo é quase o mesmo do que já fizemos anteriormente.

```
#!/usr/bin/python3
#coding: utf-8
import telepot, time
```

```
from PIL import Image
import pytesseract
```

Aqui, importamos a biblioteca do Tesseract-OCR com a linha `import pytesseract`.

Com a linha que está comentada, `telepot.api.set_proxy('http://192.168.0.1:3128', ('usuário','senha'))`, estamos preparados para configurar e operar em uma rede que passe por um proxy.

Nossa função principal ficou assim:

```
def principal(msg):
        content_type, chat_type, chat_id = telepot.glance(msg)

        if content_type == 'photo':
                caminho = ""
                imagem = msg[content_type][-1]['file_unique_id']
+ ".jpg"
                bot.download_file(msg[content_type][-1]
['file_id'], caminho + imagem)
                pytesseract.pytesseract.tesseract_cmd =
'C:\\Program Files\\Tesseract-OCR\\tesseract.exe'
                tessdata_dir_config = '--tessdata-dir
"C:\\Program Files\\Tesseract-OCR\\tessdata"'
                foto = Image.open(imagem + caminho)
                texto = pytesseract.image_to_string(foto,
lang='por', config=tessdata_dir_config)
                bot.sendMessage(chat_id,"O texto da imagem:\n" +
texto)
```

Com `if content_type == 'photo':`, verificamos se o conteúdo recebido é uma foto e, com a variável `caminho` definida como uma string vazia na linha `caminho = ""`, podemos designar um caminho específico onde o bot salvará as imagens recebidas.

Lembre-se de que devem ser caminhos válidos. Para o sistema operacional Windows, usamos barras invertidas (`\`) para dividir os diretórios. Como elas precedem caracteres especiais em Python, é necessário sempre colocá-las em duplas para serem interpretadas

como apenas uma (`\\`). Para o Linux, usamos barras normais e não há esse problema.

Com a linha `imagem = msg[content_type][-1]['file_unique_id'] + ".jpg"`, definimos que a variável `imagem` receberá o nome do arquivo definido automaticamente pelo Telegram em seus servidores e que pode ser obtido da mensagem através do vetor `content_type`. Adicionamos a extensão .jpg ao final da string para que a gravação no disco seja em um arquivo desse formato.

Já a linha `bot.download_file(msg[content_type][-1]['file_id'], caminho + imagem)` fará o download da imagem dos servidores do Telegram para o local especificado na variável `caminho` e com o nome constante na variável `imagem`.

As linhas a seguir indicam, para sistemas operacionais Windows, onde está instalado o Tesseract-OCR.

```
pytesseract.pytesseract.tesseract_cmd = 'C:\\Program
Files\\Tesseract-OCR\\tesseract.exe'
tessdata_dir_config = '--tessdata-dir "C:\\Program
Files\\Tesseract-OCR\\tessdata"'
```

O caminho mostrado aqui é o da instalação padrão. Caso você o tenha alterado na instalação do Tesseract-OCR, será necessário indicar o caminho correto aqui, tanto para o arquivo `tesseract.exe` quanto para a pasta`tessdata`.

Com `foto = Image.open(imagem + caminho)`, lemos o conteúdo do arquivo gravado para a variável `foto` para que possamos submetê-la ao Tesseract para o reconhecimento de caracteres, caso existam.

O reconhecimento dos caracteres acontece na linha `texto = pytesseract.image_to_string(foto, lang='por', config=tessdata_dir_config)`. Nesta linha, passamos a foto com a variável `foto` e, no parâmetro seguinte, a linguagem em que os caracteres estarão com `lang='por'`. O último parâmetro é o local

onde está a pasta `tessdata`, que contém os arquivos necessários para o funcionamento da rede neural de reconhecimento de caracteres.

A linha `bot.sendMessage(chat_id,"O texto da imagem:\n" + texto)`, já conhecida, enviará de volta o texto reconhecido na imagem para o usuário. Para relembrar, nas últimas linhas, indicamos o token de acesso com `bot = telepot.Bot('1409666004:AAF7eu_YN9Pu8ll0Tx_ey78NyeElJ7jUNMo')` e a função que processará as mensagens recebidas com `bot.message_loop(principal)`. O laço `while 1:` final servirá para manter o programa em execução indefinidamente.

Vamos colocar o bot para funcionar e enviar uma imagem com caracteres para testá-lo:

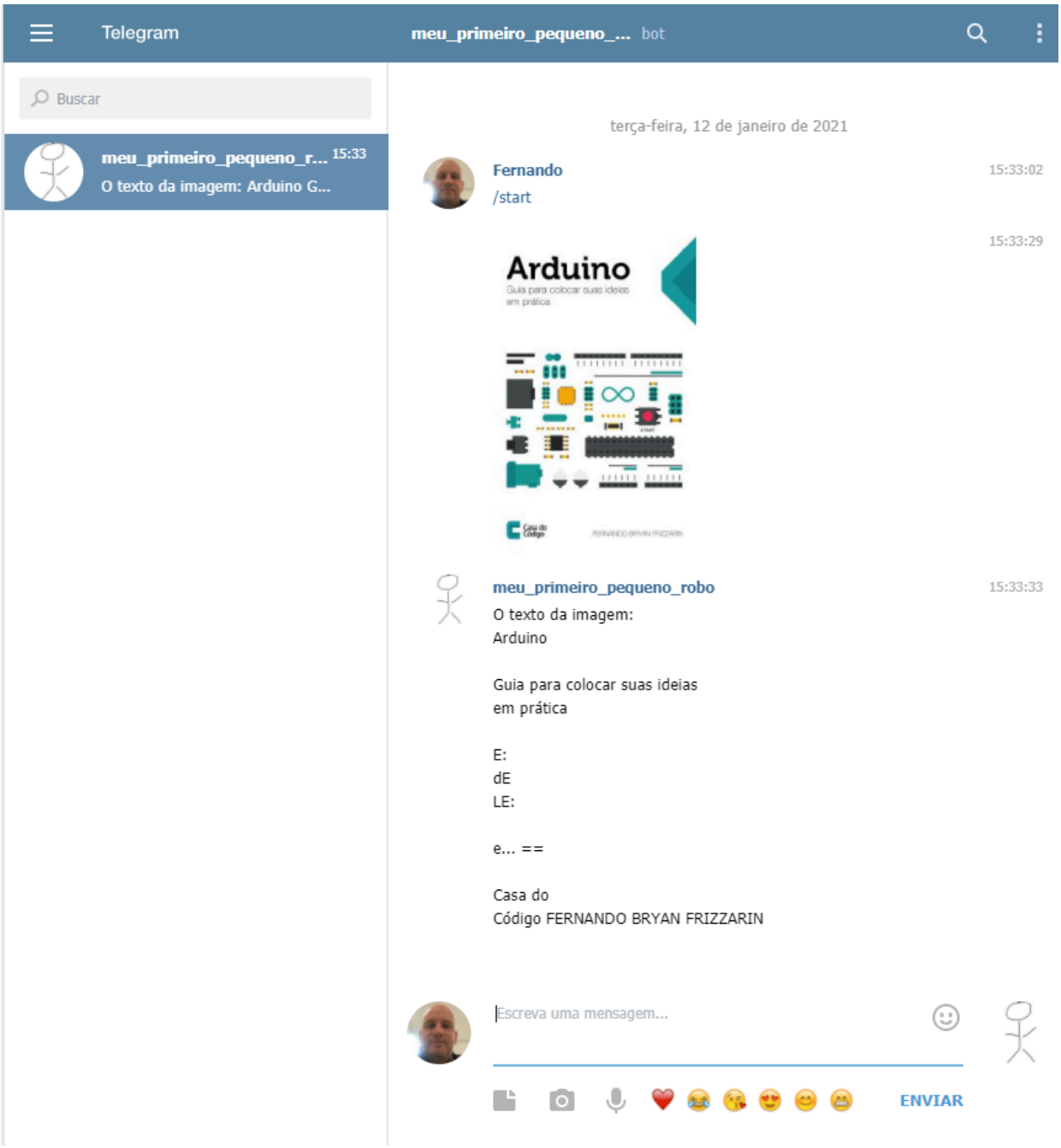


Figura 6.10: Envio de uma imagem para que o bot reconheça caracteres

Note que partes da imagem da capa do livro são reconhecidas como caracteres. Como um estudo básico, é satisfatório ter o título, o subtítulo e o nome do autor reconhecidos. Mas você pode aperfeiçoar

o reconhecimento com um bom tratamento na imagem. Tome isso como desafio para estudos posteriores.

Veja que, ao obter o texto de uma imagem, é possível proceder com inúmeros tipos de processamentos a partir disso, como realizar consultas em bancos de dados.

Por exemplo, se for uma placa de automóvel, você pode consultá-la na API do Departamento Nacional de Trânsito - DENATRAN, que está disponível em https://www.gov.br/conecta/catalogo/apis/wsdenatran. O limite é a sua imaginação!

O que será que você está imaginando aí agora? Independente do que seja, não se dê por satisfeito. O que virá a seguir pode lhe dar novas ideias ou melhorar as que você já tem!

Capítulo 7

Recebendo localização – calcular distância entre duas localidades

A analogia que costumam fazer entre a linguagem Python e um canivete suíço é realista. Isso se deve, principalmente, pelo fato de existem milhares de bibliotecas disponíveis para o Python.

Tome consciência de que vivemos em um mundo enorme e cheio de pessoas, com os mais diversos problemas possíveis, mas sempre com alguma pessoa programadora disposta a se engajar e resolver esses problemas. Nós nos valeremos justamente dessa máxima neste projeto!

Imagine que você queira saber qual é a distância entre duas localidades enviando localizações para seu chatbot no Telegram. Como a localização carrega os dados de latitude e longitude, podemos usar essas informações para realizar esse cálculo.

A distância geodésica, ou distância esférica, pode ser calculada, resumidamente, com a seguinte fórmula (disponível em https://www.ufrgs.br/lageo/calculos/dist_exp.html):

```
cos(S) = sen(latitude_origem) * sen(latitude_destino) +
cos(latitude_destino) * cos(latitude_origem) *
cos(longitude_destino – longitude_origem)
```

Felizmente, temos a biblioteca **GeoPy* para nos salvar da matemática! Ela provê uma série de funções para uso em tarefas geodésicas e de integração com outros softwares, tais como ArcGIS, AzureMaps, Google, entre outros. Uma dessas funções é justamente calcular a distância linear usando dois pontos geográficos.

Lembre-se de que não é possível apenas subtrair o destino da origem usando-se pontos geográficos. Isso porque a Terra é um geoide (esfera) e é preciso considerar sua curvatura.

Note que aplicativos e sites, tais como o Google Maps, mostram a distância através de ruas e rodovias, mas o que queremos calcular é a distância em linha reta (ou seja, a menor distância entre dois pontos) no globo terrestre, também chamada de distância geodésica.

Os comandos para instalar a biblioteca GeoPy estão logo a seguir, tanto para os sistemas operacionais Windows quanto Linux.

No Windows:

```
pip install geopy
```

No Linux:

```
pip3 install geopy
```

Com tudo pronto, começamos pelo cabeçalho, com a importação da classe `distance` da biblioteca `geopy` com a linha `from geopy import distance, location`.

Também deixamos comentada e pronta para uso a linha de proxy, caso seja necessário usá-la em sua rede.

```
#!/usr/bin/python3
#coding: utf-8
import telepot, time
from geopy import distance, location

# telepot.api.set_proxy('http://ip do proxy:porta do proxy',
('usuário','senha'))
```

Na sequência, zeramos as variáveis de início e fim de latitude e longitude. São elas que armazenarão esses respectivos dados geográficos.

```
lat_i = lat_f = lon_i = lon_f = 0
```

Para ter controle disso, como veremos no código mais à frente, se as variáveis de origem (com final `_i`) estiverem zeradas, a localização enviada é a de início; caso contrário, serão consideradas como o destino (com final `_f`).

A seguir, vem a nossa função principal para tratamento das mensagens recebidas:

```
def principal(msg):
        global lat_i, lon_i, lat_f, lon_f
        content_type, chat_type, chat_id = telepot.glance(msg)

        if content_type == 'location':
                if lat_i == 0 and lon_i == 0:
                        lat_i = msg['location']['latitude']
                        lon_i = msg['location']['longitude']
                else:
                        lat_f = msg['location']['latitude']
                        lon_f = msg['location']['longitude']
                        local_i = (lat_i, lon_i)
                        local_f = (lat_f, lon_f)
                        distancia =
distance.distance(local_i,local_f).km
                        bot.sendMessage(chat_id,"A distância
entre as duas localidades é de " + str(format(distancia,'.2f'))
+ " km.")
                        lat_i = lon_i = 0
```

Na primeira linha da função, com `global lat_i, lon_i, lat_f, lon_f`, informamos que as variáveis que receberão as latitudes e longitudes de origem (final `_i`) e destino (final `_f`) terão uso global, ou seja, poderão sofrer alterações também dentro da função, mesmo tendo sido declaradas fora dela.

Com a linha `content_type, chat_type, chat_id = telepot.glance(msg)`, obtemos os campos `content_type`, `chat_type` e `chat_id`, que são retornos do método `glance` da classe Telepot, que extrai as informações sobre a mensagem recebida.

A condição `if content_type == 'location':` filtrará apenas as mensagens que tiverem como conteúdo uma localização.

Considere o seguinte bloco:

```
if lat_i == 0 and lon_i == 0:
    lat_i = msg['location']['latitude']
    lon_i = msg['location']['longitude']
```

Se as coordenadas de latitude e longitude de origem (lembre-se, final `_i`) tiverem como conteúdo zero, isso significa que a localização enviada é a de origem e deve ser preenchida com a latitude (`lat_i = msg['location']['latitude']`) e a longitude (`lon_i = msg['location']['longitude']`) recebidas.

Porém, caso as variáveis de origem não tenham o valor zerado, quer dizer que a localização recebida é um destino (final `_f`).

```
else:
lat_f = msg['location']['latitude']
lon_f = msg['location']['longitude']
local_i = (lat_i, lon_i)
local_f = (lat_f, lon_f)
distancia = distance.distance(local_i,local_f).km
bot.sendMessage(chat_id,"A distância entre as duas localidades é
de " + str(format(distancia,'.2f')) + " km.")
lat_i = lon_i = 0
```

Nesse caso, armazenamos a latitude em `lat_f = msg['location']['latitude']` e a longitude em `lon_f = msg['location']['longitude']`.

Com essas informações, criamos uma variável para o local de origem chamada `local_i`, que armazenará uma tupla de dois campos (latitude e longitude). Fazemos o mesmo para o local de destino com a variável `local_f`.

Com a linha `distancia = distance.distance(local_i,local_f).km`, armazenamos a distância entre os dois pontos usando o método `distance`, que deve receber dois locais, de origem e destino, no formato de tupla de dois campos (latitude, longitude) – por isso criamos os locais anteriormente.

No final da linha, temos o `.km`, que faz com que o valor retornado esteja em quilômetros; caso queira em milhas, substitua-o por `.miles`.

Com a distância calculada, agora é só enviar de volta para o usuário uma mensagem com essa informação:

```
bot.sendMessage(chat_id,"A distância entre as duas localidades é de " + str(format(distancia,'.2f')) + " km.")
```

Como a variável `distancia` é do tipo número, ela precisa ser convertida em string para poder ser concatenada. Fazemos isso com a função `str()` e, como as casas decimais retornadas pelo `geopy` são muitas, usamos a função `format()` para formatá-las. A função `format` recebe o número e, na sequência, a formatação desejada. No caso, `.2f` indica que serão duas casas depois da vírgula (`float`).

Depois de a mensagem ser enviada com a distância calculada, zeramos as variáveis `lat_i` e `lon_i` (origem) para que, no caso de uma nova localização enviada, o processo todo possa ser repetido para o cálculo da distância novamente.

Para terminar, é só criar o bot passando o token de acesso e a função que tratará as mensagens, e criar um loop para que o programa permaneça rodando:

```
bot =
telepot.Bot('1510435655:AAHU99c_gxEHMtNrYmBJcQq_ao4iVRFHL4o')
bot.message_loop(principal)

while 1:
        time.sleep(5)
```

Após colocar o bot para funcionar, basta enviar duas localizações para que a distância seja calculada. Selecione a opção localização no envio da mensagem:

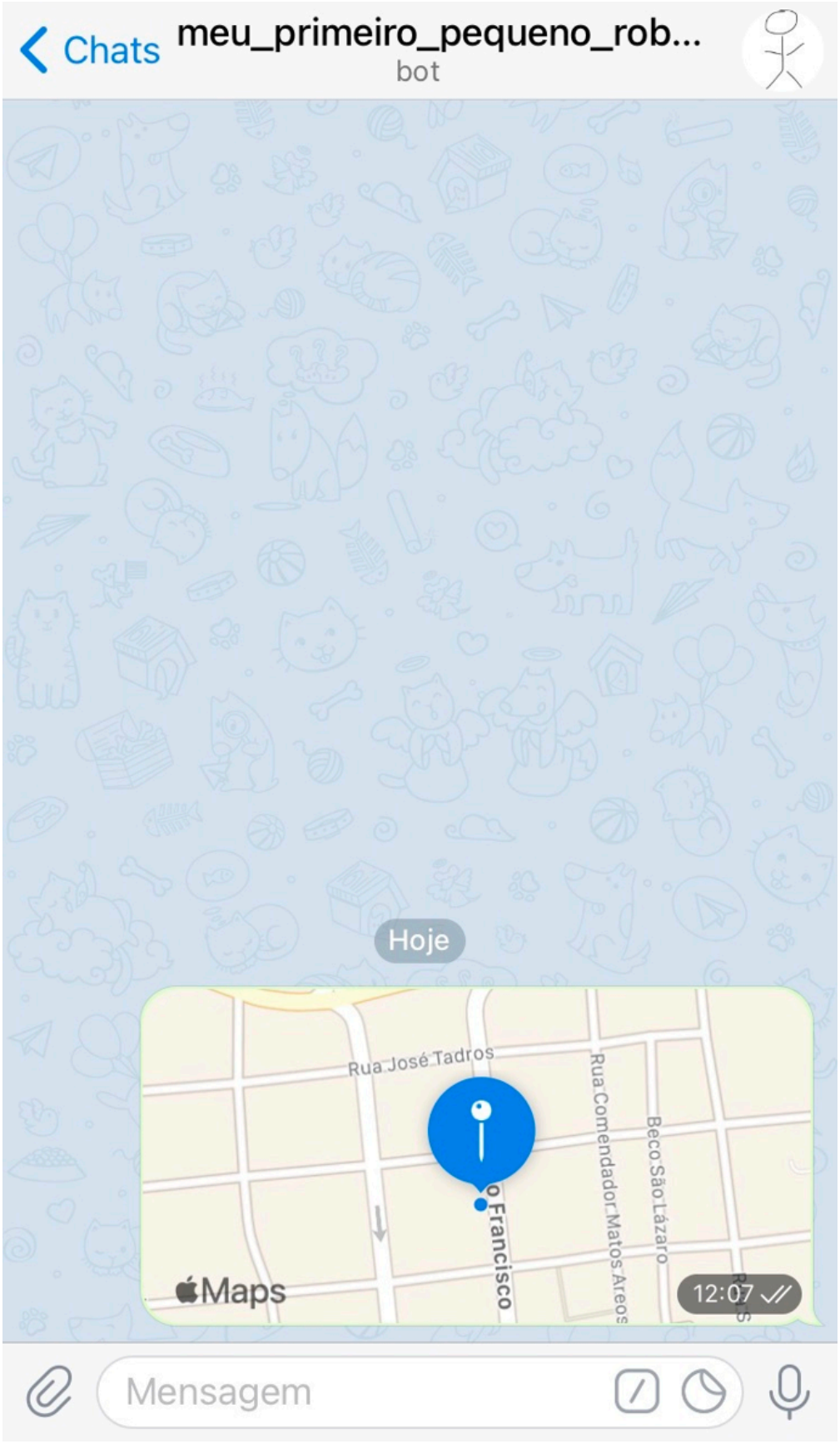

Chats
meu_primeiro_pequeno_rob...
bot
Hoje
Rua José Tadros
Rua Comendador Matos Areos
Beco São Lázaro
Maps
12:07
Mensagem

Figura 7.1: Enviando uma localização para o bot

Agora envie uma segunda localização. Quase imediatamente, o bot vai responder a distância entre as duas localizações que você enviou:

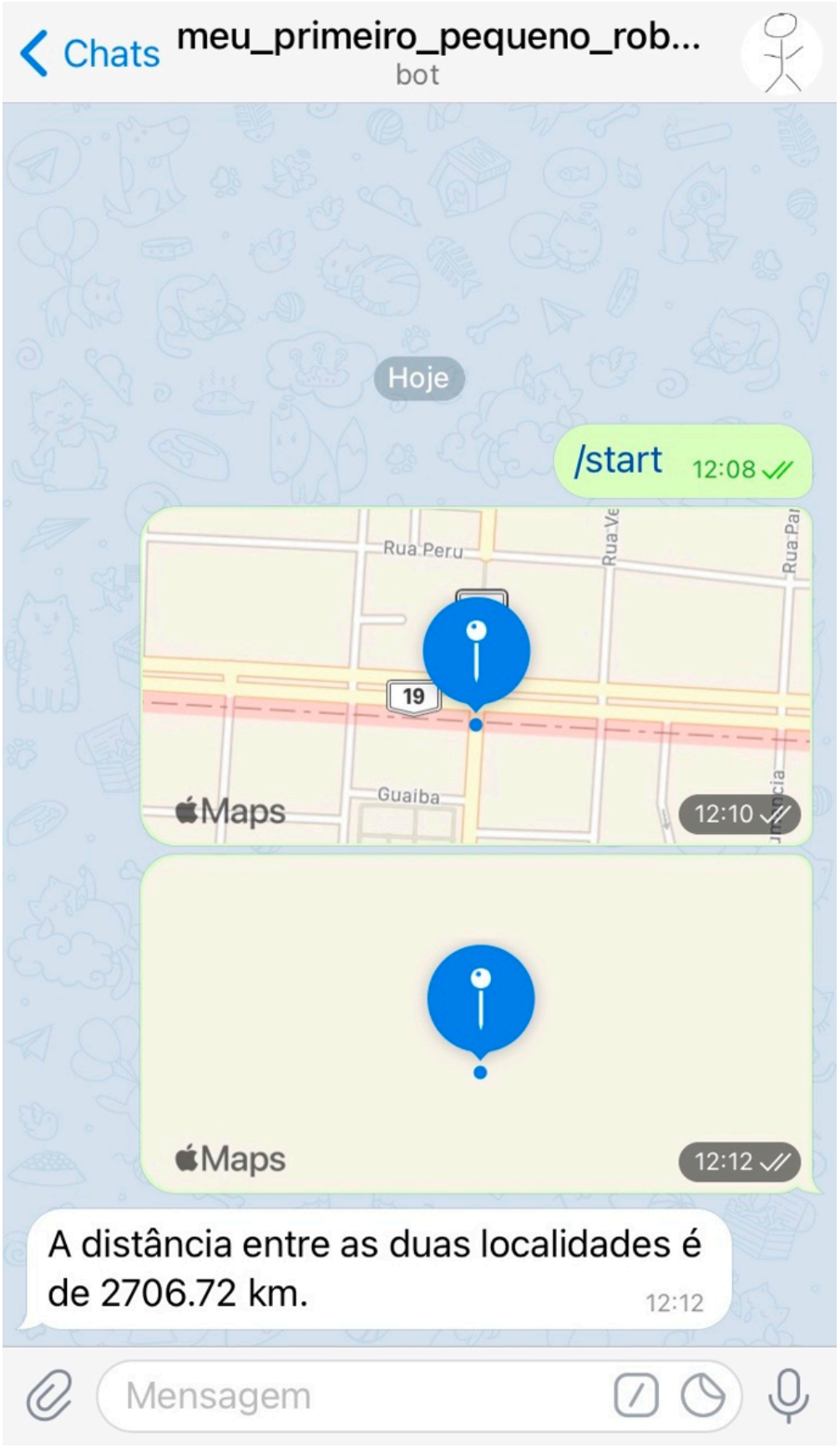

Chats
meu_primeiro_pequeno_rob...
bot
Hoje
/start 12:08
Rua Peru
19
Guaiba
Maps
12:10
Maps
12:12
A distância entre as duas localidades é de 2706.72 km.
12:12
Mensagem

Figura 7.2: Mensagem com a distância entre as duas localizações

Outra coisa que pode ser bastante interessante é o bot conseguir enviar uma localização específica para o usuário. Por exemplo, imagine que o usuário possa enviar uma mensagem para o bot solicitando a localização de uma loja ou algum local de interesse.

Para que o bot tenha essa capacidade, vamos incluir mais uma condição: a interrogação (`?`). Quando alguém enviar o caractere `?` o bot responderá com uma localização.

Primeiro, vamos ao Google Maps para escolher um lugar para definir como a localização do bot. Pode viajar à vontade! Após a escolha do local, clique com o botão direito e ele mostrará a latitude e a longitude:

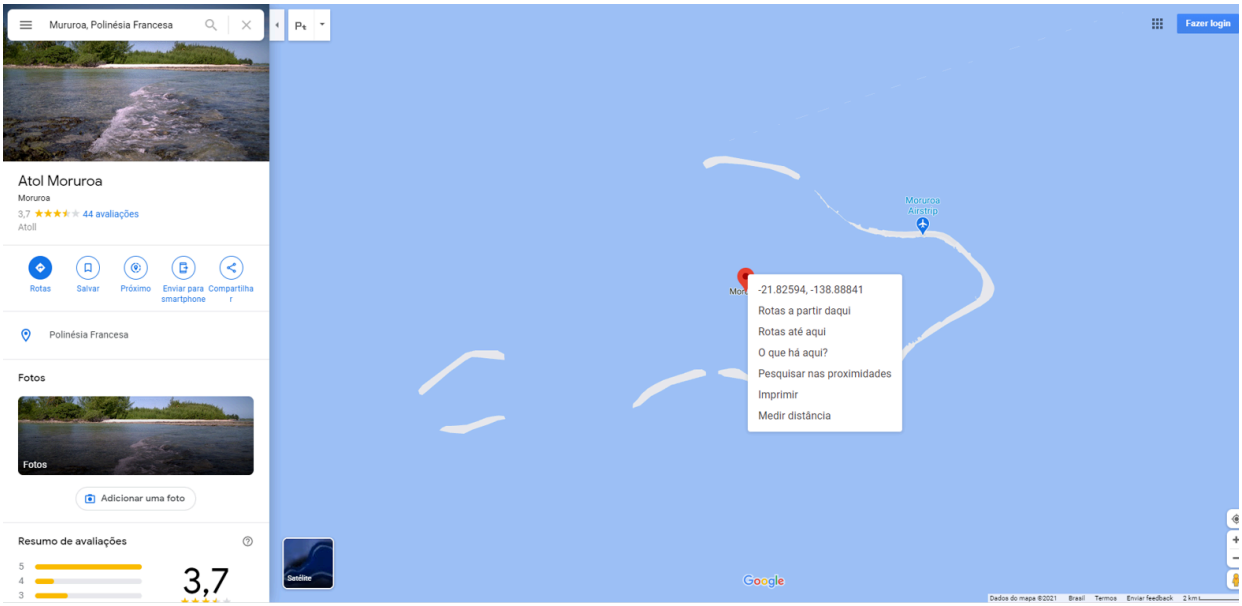


Figura 7.3: Atol de Mururoa na Polinésia Francesa

Agora vamos incrementar nosso código com o trecho para receber o caractere `?` e devolver a localização escolhida:

```
if content_type == 'text':
                chat_id = msg['chat']['id']
```

```
                mensagem = msg['text']
                if mensagem == '?':

bot.sendLocation(chat_id,'-21.82594','-138.88841')
```

A única coisa nova aqui é a linha `bot.sendLocation(chat_id,'-21.82594','-138.88841')`, onde enviamos uma localização para o `chat_id` da conversa no primeiro parâmetro. No segundo, enviamos a latitude e no terceiro, a longitude.

Como isso, o Telegram entenderá como localização geográfica e a mostrará em um mapa:

Chats
meu_primeiro_pequeno_rob...
bot
Rua Peru
19
Guaiba
Maps
12:10
Maps
12:12
A distância entre as duas localidades é de 2706.72 km.
12:12
?
12:42
Maps
12:42
Mensagem

Figura 7.4: A localização geográfica recebida no Telegram

Também é possível enviar a localização em tempo real! Basta adicionar o parâmetro `live_period` ao `sendLocation`. O parâmetro `live_period` recebe um número inteiro entre 60 e 86400, que é a quantidade de segundos em que o envio da posição geográfica em tempo real será permitido.

Para o período de uma hora, por exemplo, a linha ficará assim:

```
bot.sendLocation(chat_id,'-21.82594','-138.88841',live_period=36
00)
```

Para atualizar a posição, usa-se o método `editMessageLiveLocation`, que recebe o `chat_id`, a identificação da mensagem (`message_id`), a latitude e a longitude atuais. A identificação da mensagem é retornada toda vez que uma mensagem é enviada; portanto, basta armazenar o retorno do método `sendLocation` para obtê-la.

```
msg_id =
bot.sendLocation(chat_id,'-21.82594','-138.88841',live_period=36
00)
```

Com essa informação armazenada na variável `msg_id`, podemos usá-la para atualizar a posição geográfica em tempo real, como no exemplo:

```
bot.editMessageLiveLocation(telepot.message_identifier(msg_id),'
-15.79844','-47.86432')
```

Com `teleport.message_identifier(msg_id)` obtemos como retorno uma tupla com dois campos, onde o primeiro é o `chat_id` e o segundo, o `message_id`. Dessa forma, temos a identificação completa da mensagem que deve ser editada.

Só será possível o envio de novas posições enquanto o tempo definido em `live_period` não estiver expirado, ou usando o método `stopMessageLiveLocation`:

```
bot.stopMessageLiveLocation(telepot.message_identifier(msg_id))
```

Com isso, conseguimos parar o envio de posições em tempo real independente de o tempo de `live_period` ter ou não expirado. Vamos alterar o trecho do código anterior para este:

```
if mensagem == '?':
    # Atol de Mururoa
    msg_id =
bot.sendLocation(chat_id,'-21.82594','-138.88841',live_period=36
00)
    time.sleep(10)
    # Congresso Nacional em Brasília

bot.editMessageLiveLocation(telepot.message_identifier(msg_id),'
-15.79844','-47.86432')
    time.sleep(10)
    # Moscou, Rússia

bot.editMessageLiveLocation(telepot.message_identifier(msg_id),'
55.75560','37.61987')
    time.sleep(10)

bot.stopMessageLiveLocation(telepot.message_identifier(msg_id))
```

A cada 10 segundos, alteramos a posição da localização. Executando seu bot, conseguimos conferir as mudanças de posição no Telegram quando enviamos o caractere `?`.

A primeira posição, no Atol de Mururoa:

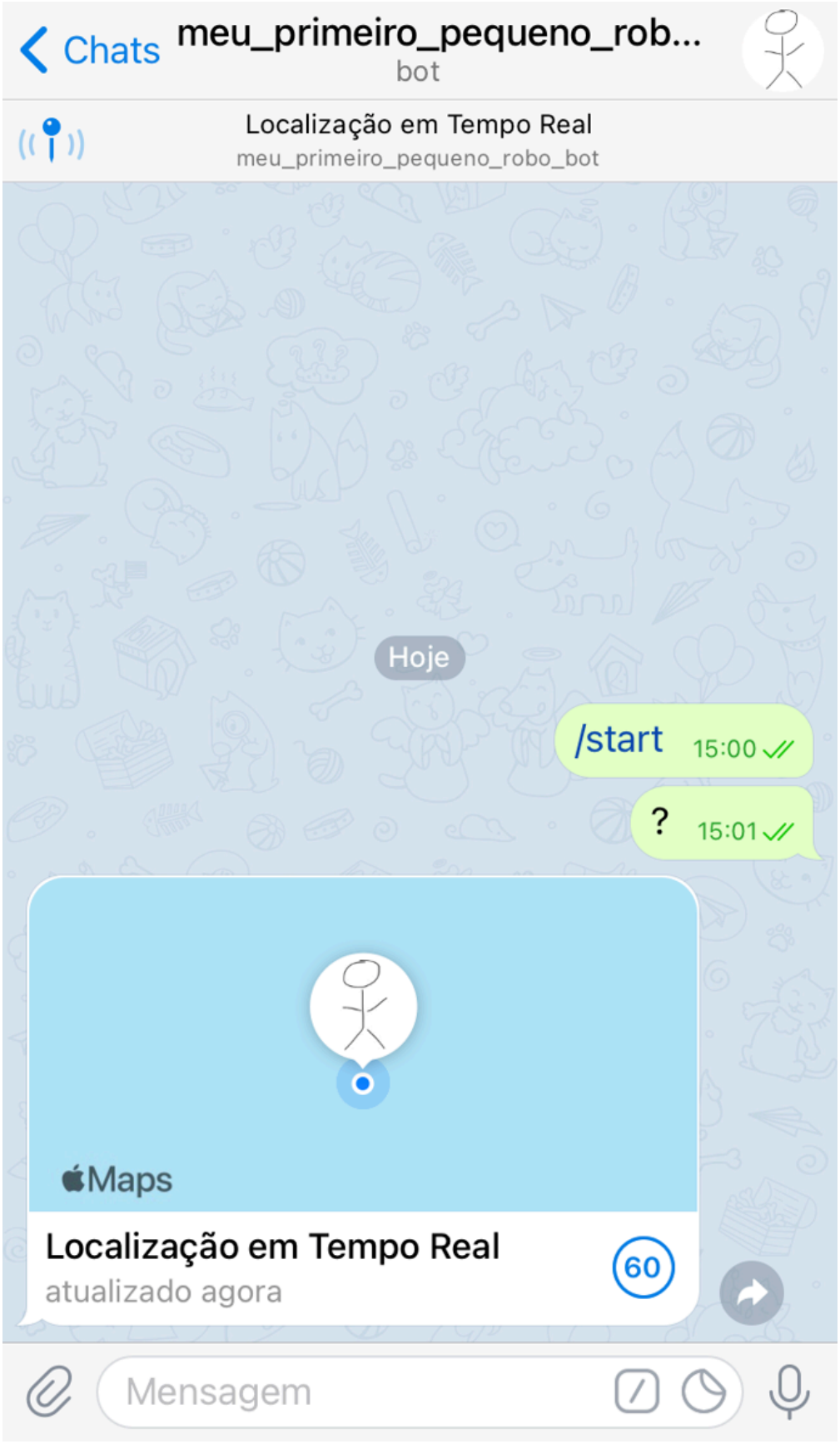

Chats
meu_primeiro_pequeno_rob...
bot
Localização em Tempo Real
meu_primeiro_pequeno_robo_bot
Hoje
/start 15:00
? 15:01
Maps
Localização em Tempo Real
atualizado agora
60
Mensagem

Figura 7.5: Localização do bot no Atol de Mururoa

A segunda posição, em Brasília:

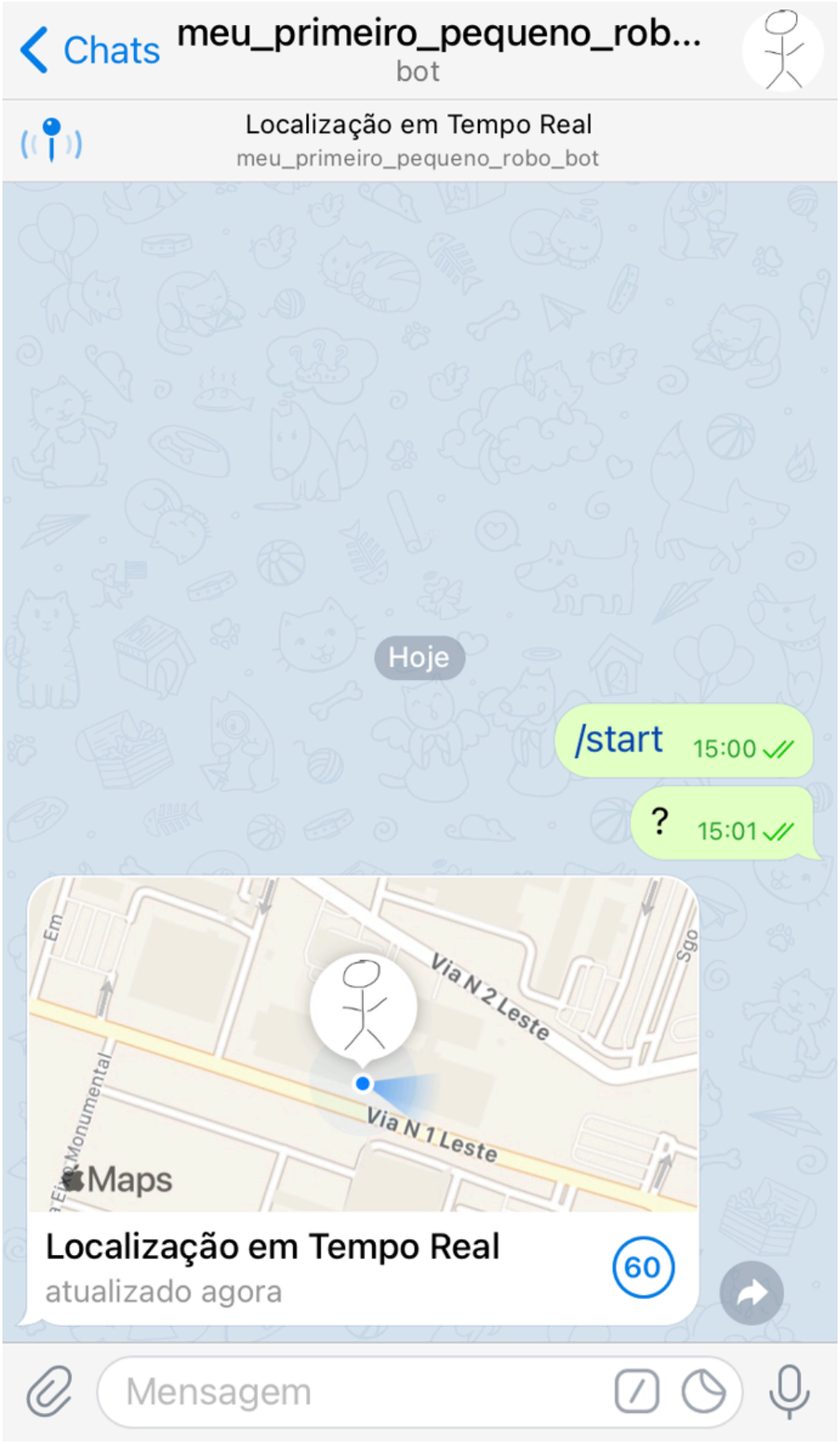

Chats
meu_primeiro_pequeno_rob...
bot
Localização em Tempo Real
meu_primeiro_pequeno_robo_bot
Hoje
/start 15:00
? 15:01
Via N 2 Leste
Via N 1 Leste
Maps
Localização em Tempo Real
atualizado agora
60
Mensagem

Figura 7.6: Localização do bot em Brasília

A terceira posição, em Moscou:

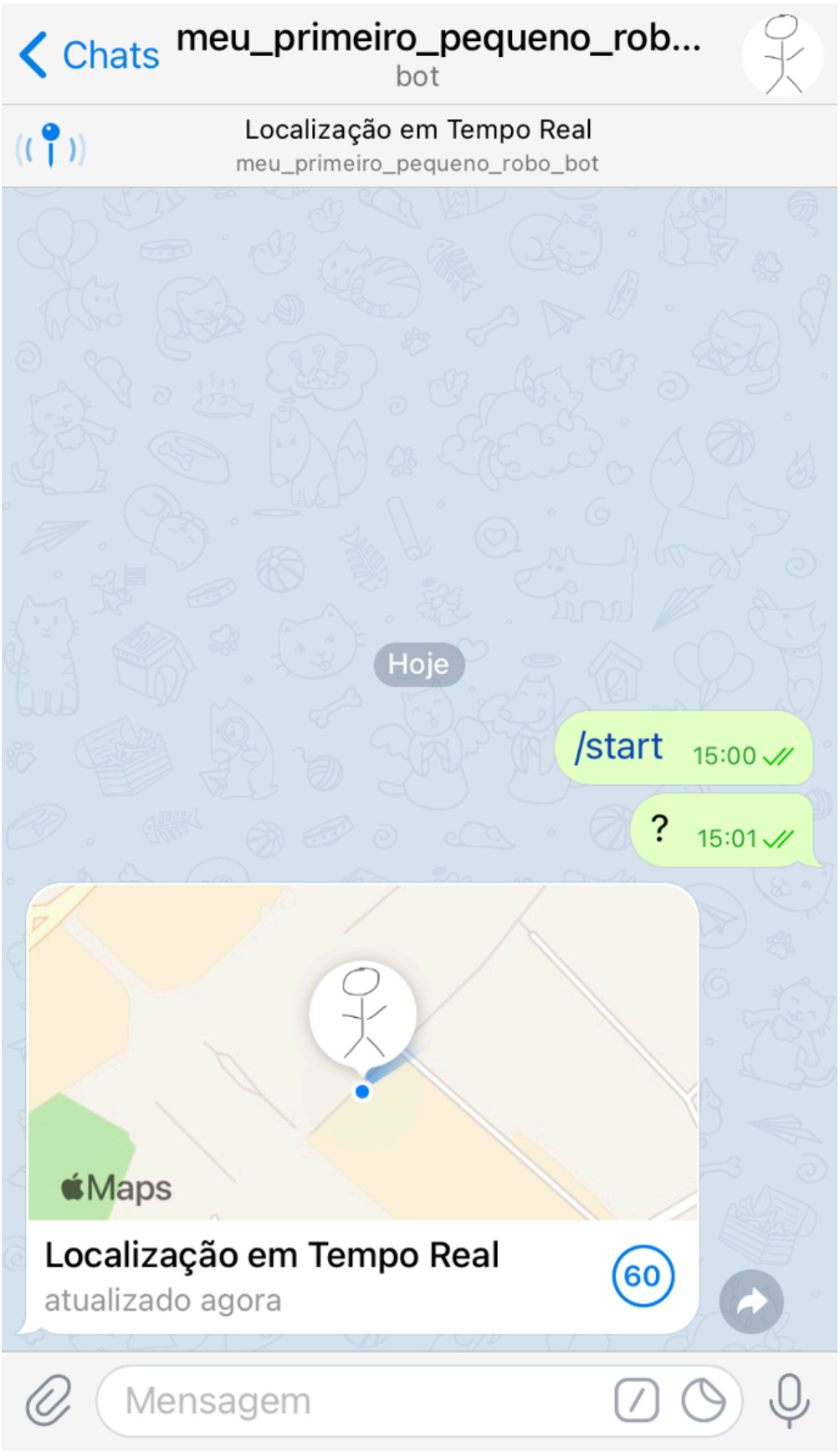

Chats
meu_primeiro_pequeno_rob...
bot
Localização em Tempo Real
meu_primeiro_pequeno_robo_bot
Hoje
/start 15:00
? 15:01
Maps
Localização em Tempo Real
atualizado agora
60
Mensagem

Figura 7.7: Localização do bot em Moscou

A primeira coisa que me vem à cabeça depois de tudo isso é um equipamento de rastreamento baseado no Telegram! Imagina um Raspberry Pi instalado em um balão, tirando fotos para baixo e enviando sua localização em tempo real para o Telegram, para que os dados possam ser recuperados depois?!

Ou imagine o cliente de uma loja perguntando onde ela fica e recebendo a posição geográfica do local para abrir no GPS no seu telefone celular e ter uma rota imediata até lá? Bom, mas isso daria outro livro! Quem sabe será você a escrevê-lo?

Vamos em frente, como sempre! Que tal um bot para jogar no Telegram?

Capítulo 8

Jogo da velha - Usando menus para interação com usuário

Uma das coisas interessantes do Telegram é a possibilidade de criar e usar menus para interação com o usuário. Isso permite que os chatbots apresentem opções possíveis e não tenham que tratar mensagens com as suas mais variadas possibilidades de texto.

Você já viu esses menus que o Telegram disponibiliza! Lembra-se quando interagimos com o @BotFather? Veja na imagem a seguir:

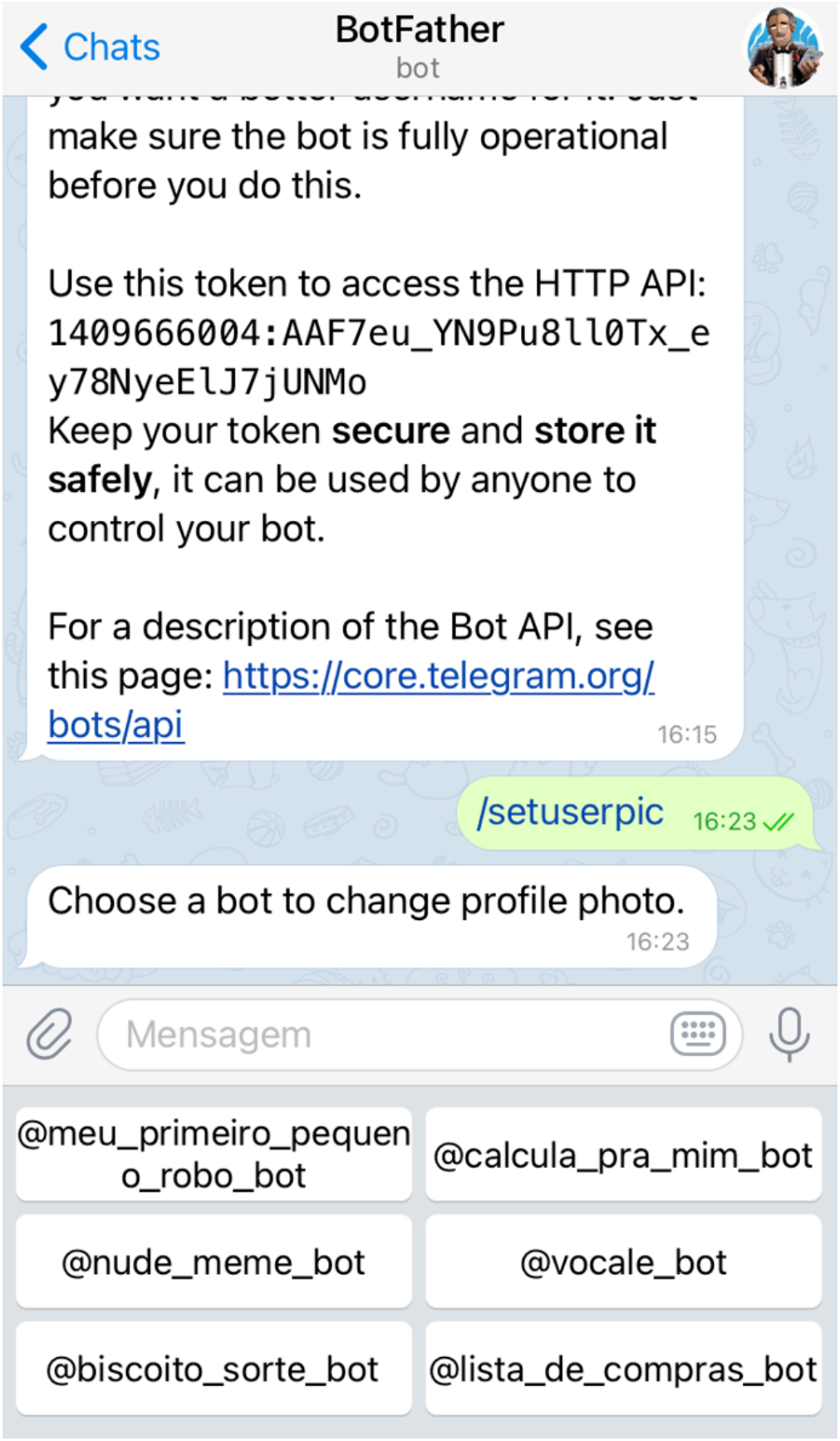

Chats
BotFather
bot
make sure the bot is fully operational before you do this.
Use this token to access the HTTP API:
1409666004:AAF7eu_YN9Pu8ll0Tx_ey78NyeElJ7jUNMo
Keep your token secure and store it safely, it can be used by anyone to control your bot.
For a description of the Bot API, see this page: https://core.telegram.org/bots/api
16:15
/setuserpic
16:23
Choose a bot to change profile photo.
16:23
Mensagem
@meu_primeiro_pequeno_robo_bot
@calcula_pra_mim_bot
@nude_meme_bot
@vocale_bot
@biscoito_sorte_bot
@lista_de_compras_bot

Figura 8.1: Exemplo de menu @BotFather

Onde estão os nomes dos bots, abaixo da área de digitação, está um menu. Com eles, podemos direcionar o usuário para que ele selecione apenas uma das opções possíveis.

O valor da opção selecionada será enviado para o Telegram, na conversa, como texto. Ou seja, ao clicar no botão `@meu_primeiro_pequeno_robo_bot`, exatamente esse texto, que contém o nome do bot, será enviado para a conversa.

Isso facilita muito na hora de tratar as mensagens. Você verá que não será necessário fazer nada muito diferente do que já fizemos até agora, só acrescentaremos o menu e suas opções. Então, chega de conversa, vamos colocar a mão na massa!

A ideia deste capítulo é usar essa possibilidade de criar menus para concretizarmos um jogo da velha.

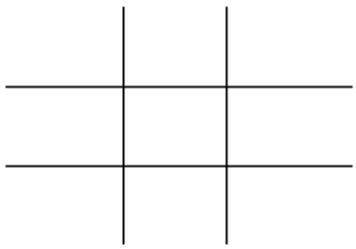


Figura 8.2: O tabuleiro do jogo da velha

Como não conseguiríamos desenhar o tabuleiro da imagem anterior e fazer com que o usuário clique sobre ele, vamos usar um menu com três linhas e cada linha terá três botões, que indicam as posições possíveis de serem jogadas.

Dessa forma, disponibilizaremos posições de 1 até 9 para serem jogadas, como no esquema da imagem a seguir.

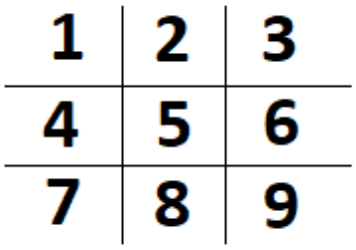


Figura 8.3: O "tabuleiro" do jogo da velha com a numeração de cada posição de jogada

Consideraremos que o usuário será o "X" e o computador o "O".

No nosso código, vamos incluir duas novas bibliotecas para importação, mas não serão necessárias novas instalações. Vamos importar da biblioteca **telepot** a classe `_namedtuple_`. Essa classe possibilita espelhar os objetos criados para os bots do Telepot. Em algumas situações, queremos completar um objeto com valores que são "nada" (`none`). Essa conversão faz com que seja possível utilizarmos todos os campos dos objetos e preenchê-los, mesmo que seja com "nada" (`none`).

Dessa classe, ainda importaremos dois objetos:`ReplyKeyboardMarkup` e `KeyboardButton`. O primeiro permite que formatemos texto para os botões e o segundo permite a criação efetiva dos botões.

Com isso, poderemos criar menus da seguinte forma:

```
bot.sendMessage(chat_id,"Selecione uma opção:",
reply_markup=ReplyKeyboardMarkup(keyboard=opcoes,
resize_keyboard = True, one_time_keyboard = True))
```

O seguinte trecho criará o menu com seus botões:

```
reply_markup=ReplyKeyboardMarkup(keyboard=opcoes,
resize_keyboard = True, one_time_keyboard = True).
```

Os botões do menu serão organizados em forma de matriz com a quantidade de linhas e colunas coincidentes, conforme a declaração

desta variável ocorrer no código-fonte, o qual veremos adiante.

A opção `resize_keyboard = True` permite que o menu seja ajustado automaticamente conforme a resolução da tela. Já a opção `one_time_keyboard = True` fará com que o menu seja mostrado apenas uma vez e depois ocultado.

Bem, já adiantamos muito o expediente do que temos de novidade neste capítulo. Vamos destrinchar o código-fonte todo:

```
#!/usr/bin/python3
#coding: utf-8
import telepot, time
from telepot.namedtuple import ReplyKeyboardMarkup,
KeyboardButton
from random import randint
```

Sobre a `namedtuple`, é o que já foi mencionado anteriormente. A biblioteca `random` provê comandos e funções para criação e manipulação de números aleatórios. Com o `randint`, podemos criar números aleatórios inteiros.

Precisamos criar uma variável para conter as opções do menu. Lembre-se, serão as posições do jogo da velha. Ela estará no começo porque terá uso geral pelo programa e, uma vez criada, apenas será modificada a cada jogada. Faremos isso com:

```
opcoes = []
```

Para a função de recebimento e tratamento das mensagens, teremos aquele começo tradicional que fizemos até agora:

```
def principal(msg):
        global opcoes
        content_type, chat_type, chat_id = telepot.glance(msg)

        if content_type == "text":
                chat_id = msg["chat"]["id"]
                mensagem = msg["text"]
```

Já na sequência, vamos tratar as mensagens que indicarão o início das jogadas. Escolhemos as palavras "JOGAR" e "SIM" – "jogar", por motivos óbvios, mas o "sim" é porque, no final, perguntaremos se o usuário quer jogar novamente, com um menu de "sim ou não".

Uma vez dado o comando para início de jogo, preenchemos a variável `opcoes` com as posições de jogadas.

```
if mensagem.upper() == "JOGAR" or mensagem.upper() == "SIM":
                    opcoes = [["1","2","3"],["4","5","6"],
["7","8","9"]]
```

Lembre-se de que o menu do Telegram é uma matriz, então `opcoes` representa uma matriz 3x3 com a seguinte distribuição:

```
1    2    3
4    5    6
7    8    9
```

Feito isso, é só enviar o menu como uma mensagem para o usuário. Faremos isso com a linha:

```
bot.sendMessage(chat_id,"Selecione uma opção:",
reply_markup=ReplyKeyboardMarkup(keyboard=opcoes,
resize_keyboard = True, one_time_keyboard = True))
```

Como essa linha foi detalhada anteriormente, agora é só tratar as jogadas.

O usuário vai clicar em um botão do menu com um número correspondente para jogar. Como o conteúdo do botão será enviado como mensagem, ele também poderá digitar um número de 1 a 9.

Para receber isso, não precisaremos criar uma estrutura de seleção `if`. Para "pegar" as jogadas, usaremos a estrutura `try: ... exception: ...`, ou seja, tentaremos fazer alguma coisa e, se der errado, o código que será executado é o que estiver na cláusula `exception`. Se for tudo bem, aí trataremos a mensagem.

Faremos isso com essas duas linhas:

```
try:
    if int(mensagem) in range(0,10):
```

Note que com `int(mensagem)` tentaremos converter a mensagem em número. Se ela não for um número, teremos uma mensagem de erro aqui. Como ela está dentro da cláusula `try`, com esse erro, o programa passará a executar o que estiver na cláusula `exception`, que virá mais adiante.

Para verificar se há mensagem, usamos a estrutura `try: ... exception: ...`. Agora, para verificar as jogadas, usaremos a estrutura de seleção `if`. Dessa forma, saberemos em qual posição da matriz do menu foi a jogada, o que substituirá o número da posição presente nela pelo caractere `X`. Com isso, temos a jogada do usuário.

```
if mensagem == "1": opcoes[0][0] = "X"
if mensagem == "2": opcoes[0][1] = "X"
if mensagem == "3": opcoes[0][2] = "X"
if mensagem == "4": opcoes[1][0] = "X"
if mensagem == "5": opcoes[1][1] = "X"
if mensagem == "6": opcoes[1][2] = "X"
if mensagem == "7": opcoes[2][0] = "X"
if mensagem == "8": opcoes[2][1] = "X"
if mensagem == "9": opcoes[2][2] = "X"
```

Com a jogada do usuário feita, vamos sortear aleatoriamente uma posição para ser a jogada do programa:

```
l = randint(0,2)
c = randint(0,2)
```

Com `randint(0,2)` sorteamos um número aleatório entre 0 e 2.

Mas, veja: como não controlamos que números podem ser sorteados para a linha (variável `l`) ou para a coluna (variável `c`), elas podem indicar alguma posição onde já temos uma jogada.

Para evitar isso, usaremos um laço de repetição `while`. Enquanto o número sorteado estiver em alguma posição que já tenha alguma

jogada, ele sorteará novamente os números para a linha e para a coluna.

```
cont = 0
while opcoes[l][c] == "X" or opcoes[l][c] == "O" and cont < 9:
l = randint(0,2)
c = randint(0,2)
cont = cont + 1
```

E como o tabuleiro pode já estar completo, criamos a variável `cont`, que é inicializada com 0 e pode ser acumulada até 9. Ou seja, esse trecho poderá sortear até 9 vezes para encontrar uma posição livre; caso contrário, continuará a execução sem entrar em um loop infinito.

Essa quantidade 9 foi obtida pela quantidade de posições possíveis no jogo da velha, ou seja, 3 linhas vezes 3 colunas.

Com a posição de jogada do programa sorteada corretamente, será necessário incluí-la na matriz `opcoes`. Faremos isso com:

```
opcoes[l][c] = "O"
```

Ainda antes de terminar, precisaremos verificar se alguém ganhou!

A variável `fim`, inicializada com zero, com a linha `fim = 0`, nos indicará quem ganhou. Se `fim` for igual a 1, indicará que o usuário ganhou; se `fim` for igual a 2, o vencedor será o programa. Caso ocorra um empate, o desafiante ganha – nesse caso o programa. ;)

```
fim = 0
# - Usuário ganha
if opcoes[0][0] == "X" and opcoes[0][1] == "X" and opcoes[0][2]
== "X": fim = 1
if opcoes[1][0] == "X" and opcoes[1][1] == "X" and opcoes[1][2]
== "X": fim = 1
if opcoes[2][0] == "X" and opcoes[2][1] == "X" and opcoes[2][2]
== "X": fim = 1
if opcoes[0][0] == "X" and opcoes[1][0] == "X" and opcoes[2][0]
== "X": fim = 1
if opcoes[0][1] == "X" and opcoes[1][1] == "X" and opcoes[2][1]
== "X": fim = 1
```

```
if opcoes[0][2] == "X" and opcoes[1][2] == "X" and opcoes[2][2]
== "X": fim = 1
if opcoes[0][0] == "X" and opcoes[1][1] == "X" and opcoes[2][2]
== "X": fim = 1
if opcoes[0][2] == "X" and opcoes[1][1] == "X" and opcoes[2][0]
== "X": fim = 1
# - Programa ganha
if opcoes[0][0] == "O" and opcoes[0][1] == "O" and opcoes[0][2]
== "O": fim = 2
if opcoes[1][0] == "O" and opcoes[1][1] == "O" and opcoes[1][2]
== "O": fim = 2
if opcoes[2][0] == "O" and opcoes[2][1] == "O" and opcoes[2][2]
== "O": fim = 2
if opcoes[0][0] == "O" and opcoes[1][0] == "O" and opcoes[2][0]
== "O": fim = 2
if opcoes[0][1] == "O" and opcoes[1][1] == "O" and opcoes[2][1]
== "O": fim = 2
if opcoes[0][2] == "O" and opcoes[1][2] == "O" and opcoes[2][2]
== "O": fim = 2
if opcoes[0][0] == "O" and opcoes[1][1] == "O" and opcoes[2][2]
== "O": fim = 2
if opcoes[0][2] == "O" and opcoes[1][1] == "O" and opcoes[2][0]
== "O": fim = 2
```

As estruturas de seleção `if` são autoexplicativas, mas veja na imagem a seguir uma possibilidade de ganhador:

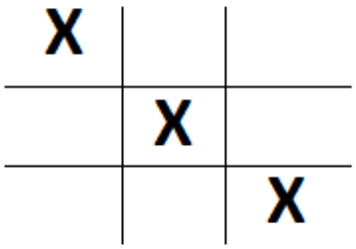


Figura 8.4: Uma possibilidade de vencedor

No caso dessa imagem, veja que, considerando o tabuleiro uma matriz, temos as posições (linha,coluna) preenchidas com "X": (0,0), (1,1) e (2,2), que indicam vitória do usuário.

No caso da imagem a seguir, as posições (2,0), (2,1) e (2,2) estão preenchidas com "O", o que indica vitória do programa.

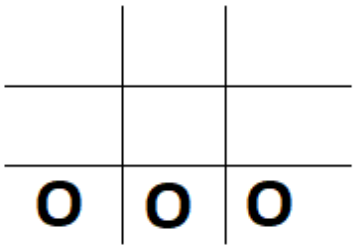


Figura 8.5: Programa ganha

Uma vez que o vencedor do jogo já tenha sido detectado, basta verificar o conteúdo da variável `fim` para saber quem ganhou. Caso a variável `fim` não tenha sido alterada, o jogo seguirá normalmente.

Tudo isso será feito com o seguinte bloco de programação:

```
if fim == 0:
    bot.sendMessage(chat_id,"Selecione uma posição:",
reply_markup=ReplyKeyboardMarkup(keyboard=opcoes,
resize_keyboard = True, one_time_keyboard = True))
    if fim == 1 or fim == 2:
        if fim == 1:
            bot.sendMessage(chat_id,"Parabéns! Você ganhou!")
            if fim == 2:
                bot.sendMessage(chat_id,"Legal! Eu ganhei!")
                jogo = ""
                for i in range(0,3):
                    for j in range(0,3):
                        jogo = jogo + opcoes[i][j] + "\t"
                    jogo = jogo + "\n"
                bot.sendMessage(chat_id,jogo)
                opcoes = [["Sim","Não"]]
                bot.sendMessage(chat_id,"Jogar novamente?",
reply_markup=ReplyKeyboardMarkup(keyboard=opcoes,
resize_keyboard = True, one_time_keyboard = True))
```

Lembre-se: se a variável `fim` for igual a 1, ganha o usuário; se for igual a 2, ganha o programa. Isso é feito com o seguinte trecho:

```
if fim == 1:
    bot.sendMessage(chat_id,"Parabéns! Você ganhou!")
    if fim == 2:
        bot.sendMessage(chat_id,"Legal! Eu ganhei!")
```

Com o trecho a seguir, criamos uma identificação visual do ganhador, que fará com que a mensagem final seja exibida:

```
for i in range(0,3):
    for j in range(0,3):
        jogo = jogo + opcoes[i][j] + "\t"
    jogo = jogo + "\n"
```

A mensagem final de vencedor da rodada é exibida como na figura a seguir:

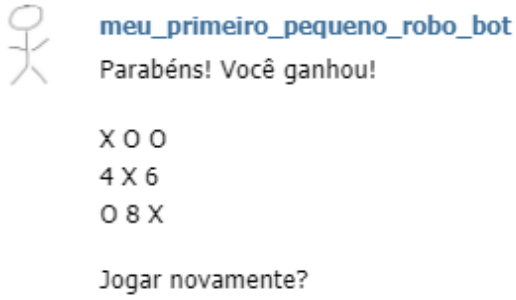


Figura 8.6: Mensagem final de vencedor

Nas posições onde estão números, em vez de `X` ou `O`, estão as posições que não receberam jogadas.

Para terminar, temos essas duas linhas com as quais fazemos o menu de jogar novamente, "sim" ou "não".

```
opcoes = [["Sim","Não"]]
bot.sendMessage(chat_id,"Jogar novamente?",
reply_markup=ReplyKeyboardMarkup(keyboard=opcoes,
resize_keyboard = True, one_time_keyboard = True))
```

Se for "sim", essa mensagem será capturada lá no início e o jogo recomeçará. Se for "não", nada muda. O código completo ficará assim:

```
#!/usr/bin/python3
#coding: utf-8
import telepot, time
from telepot.namedtuple import ReplyKeyboardMarkup,
KeyboardButton
```

```
from random import randint

# telepot.api.set_proxy('http://192.168.0.1:3128',
(usuário,senha))

opcoes = []

def principal(msg):
        global opcoes
        content_type, chat_type, chat_id = telepot.glance(msg)

        if content_type == "text":
                chat_id = msg["chat"]["id"]
                mensagem = msg["text"]

                if mensagem.upper() == "JOGAR" or
mensagem.upper() == "SIM":
                    opcoes = [["1","2","3"],["4","5","6"],
["7","8","9"]]
                    bot.sendMessage(chat_id,"Selecione uma
opção:", reply_markup=ReplyKeyboardMarkup(keyboard=opcoes,
resize_keyboard = True, one_time_keyboard = True))

                try:
                    if int(mensagem) in range(0,10):
                        if mensagem == "1": opcoes[0][0] = "X"
                        if mensagem == "2": opcoes[0][1] = "X"
                        if mensagem == "3": opcoes[0][2] = "X"
                        if mensagem == "4": opcoes[1][0] = "X"
                        if mensagem == "5": opcoes[1][1] = "X"
                        if mensagem == "6": opcoes[1][2] = "X"
                        if mensagem == "7": opcoes[2][0] = "X"
                        if mensagem == "8": opcoes[2][1] = "X"
                        if mensagem == "9": opcoes[2][2] = "X"
                        l = randint(0,2)
                        c = randint(0,2)
                        cont = 0
                        while opcoes[l][c] == "X" or opcoes[l]
[c] == "O" and cont < 9:
                            l = randint(0,2)
                            c = randint(0,2)
                            cont = cont + 1
                        opcoes[l][c] = "O"
                        fim = 0
                        # - Usuário ganha
                        if opcoes[0][0] == "X" and opcoes[0][1]
== "X" and opcoes[0][2] == "X": fim = 1
                        if opcoes[1][0] == "X" and opcoes[1][1]
```

```
== "X" and opcoes[1][2] == "X": fim = 1
                        if opcoes[2][0] == "X" and opcoes[2][1]
== "X" and opcoes[2][2] == "X": fim = 1
                        if opcoes[0][0] == "X" and opcoes[1][0]
== "X" and opcoes[2][0] == "X": fim = 1
                        if opcoes[0][1] == "X" and opcoes[1][1]
== "X" and opcoes[2][1] == "X": fim = 1
                        if opcoes[0][2] == "X" and opcoes[1][2]
== "X" and opcoes[2][2] == "X": fim = 1
                        if opcoes[0][0] == "X" and opcoes[1][1]
== "X" and opcoes[2][2] == "X": fim = 1
                        if opcoes[0][2] == "X" and opcoes[1][1]
== "X" and opcoes[2][0] == "X": fim = 1
                        # - Programa ganha
                        if opcoes[0][0] == "O" and opcoes[0][1]
== "O" and opcoes[0][2] == "O": fim = 2
                        if opcoes[1][0] == "O" and opcoes[1][1]
== "O" and opcoes[1][2] == "O": fim = 2
                        if opcoes[2][0] == "O" and opcoes[2][1]
== "O" and opcoes[2][2] == "O": fim = 2
                        if opcoes[0][0] == "O" and opcoes[1][0]
== "O" and opcoes[2][0] == "O": fim = 2
                        if opcoes[0][1] == "O" and opcoes[1][1]
== "O" and opcoes[2][1] == "O": fim = 2
                        if opcoes[0][2] == "O" and opcoes[1][2]
== "O" and opcoes[2][2] == "O": fim = 2
                        if opcoes[0][0] == "O" and opcoes[1][1]
== "O" and opcoes[2][2] == "O": fim = 2
                        if opcoes[0][2] == "O" and opcoes[1][1]
== "O" and opcoes[2][0] == "O": fim = 2
                        if fim == 0:
                            bot.sendMessage(chat_id,"Selecione
uma posição:", reply_markup=ReplyKeyboardMarkup(keyboard=opcoes,
resize_keyboard = True, one_time_keyboard = True))
                        if fim == 1 or fim == 2:
                            if fim == 1:

bot.sendMessage(chat_id,"Parabéns! Você ganhou!")
                            if fim == 2:
                                bot.sendMessage(chat_id,"Legal!
Eu ganhei!")
                            jogo = ""
                            for i in range(0,3):
                                for j in range(0,3):
                                    jogo = jogo + opcoes[i][j] +
"\t"
                                jogo = jogo + "\n"
                            bot.sendMessage(chat_id,jogo)
```

```
                            opcoes = [["Sim","Não"]]
                            bot.sendMessage(chat_id,"Jogar
novamente?", reply_markup=ReplyKeyboardMarkup(keyboard=opcoes,
resize_keyboard = True, one_time_keyboard = True))
                except ValueError:
                    pass


bot =
telepot.Bot('1510435655:AAHU99c_gxEHMtNrYmBJcQq_ao4iVRFHL4o')
bot.message_loop(principal)

while 1:
        time.sleep(5)
```

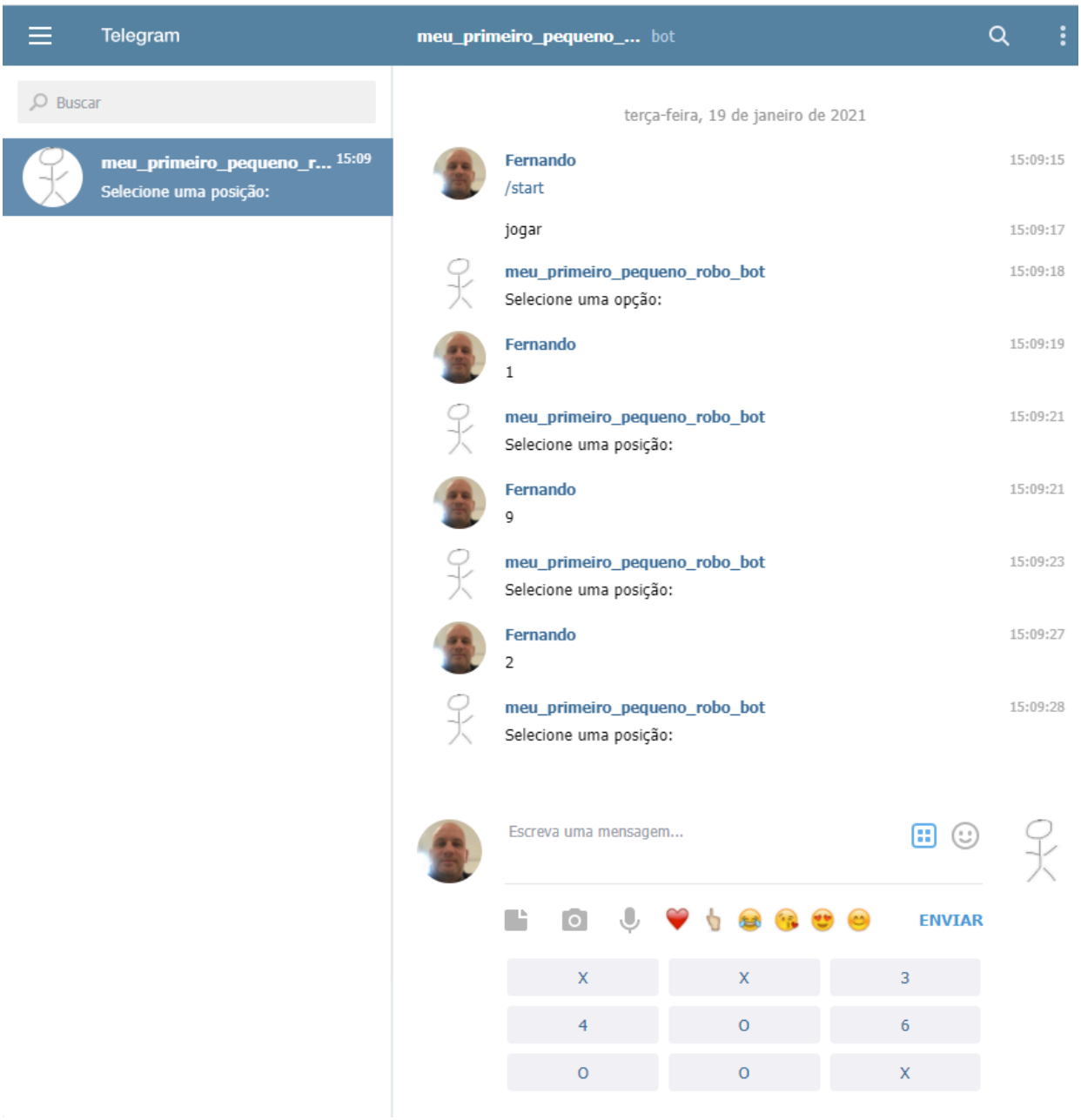


Figura 8.7: Jogando o jogo da velha pelo chatbot

Agora é só diversão!

Bom, acho que não. Eu ainda não estou satisfeito com o resultado. E você?

Acho que seria mais interessante se tivéssemos um placar que contabilizasse as vitórias do usuário e do programa. Que tal?

Há várias maneiras diferentes de fazer isso. Uma delas é criar um arquivo texto com essas informações; a outra é usar um banco de dados. Como o banco de dados será o assunto do próximo capítulo (aguenta aí, vamos terminar este antes de correrrmos para lá), por ora usaremos arquivos texto.

A lógica será a seguinte: vamos criar um arquivo texto que terá como nome o identificador `chat_id`, que são números. Nele, teremos duas linhas, a primeira para o número de vitórias do usuário e a segunda, para as vitórias do programa.

Como o nome do arquivo será o `chat_id`, será muito fácil guardar as informações no arquivo correto e recuperá-las da mesma maneira.

Para isso, vamos colocar toda essa manipulação de arquivo dentro de `if fim == 1 or fim == 2:`, que controla quem ganhou a partida. Alteraremos esse bloco para o que segue:

```
if fim == 1 or fim == 2:
    conteudo = list()
    try:
        arquivo = open(str(chat_id) + ".txt","r")
        conteudo.append(arquivo.readline().replace("\n",""))
        conteudo.append(arquivo.readline())
    except FileNotFoundError:
        conteudo.append("0")
        conteudo.append("0")
    if fim == 1:
        conteudo[0] = str(int(conteudo[0]) + 1)
        bot.sendMessage(chat_id,"Parabéns! Você ganhou!")
    if fim == 2:
        conteudo[1] = str(int(conteudo[1]) + 1)
        bot.sendMessage(chat_id,"Legal! Eu ganhei!")
    arquivo = open(str(chat_id) + ".txt","w")
    arquivo.write(conteudo[0] + "\n")
    arquivo.write(conteudo[1])
    arquivo.close()
    jogo = ""
    print(conteudo)
    for i in range(0,3):
        for j in range(0,3):
            jogo = jogo + opcoes[i][j] + "\t"
            jogo = jogo + "\n"
```

```
    bot.sendMessage(chat_id,jogo)

    opcoes = [["Sim","Não"]]
    bot.sendMessage(chat_id,"Jogar novamente?",
reply_markup=ReplyKeyboardMarkup(keyboard=opcoes,
resize_keyboard = True, one_time_keyboard = True))
```

Com `conteudo = list()`, criamos a variável `conteudo`, que será uma lista. Com isso, conseguiremos incluir e obter o conteúdo dela através de índices, como um vetor.

Agora, vejamos o seguinte bloco:

```
try:
    arquivo = open(str(chat_id) + ".txt","r")
    conteudo.append(arquivo.readline().replace("\n",""))
    conteudo.append(arquivo.readline())
except FileNotFoundError:
    conteudo.append("0")
    conteudo.append("0")
```

Com `arquivo = open(str(chat_id) + ".txt","r")`, abrimos o arquivo que tem como nome o número do `chat_id` e extensão `.txt`. Ele contém o placar apenas para leitura, indicado pelo parâmetro "r".

Esse parâmetro pode ser:

Caractere	**Descrição**
r	Somente leitura
w	Escrita, se o arquivo existir, o substitui
x	Escrita, retorna erro caso o arquivo já exista
a	Escrita, adiciona novo conteúdo ao final
b	Modo binário
t	Modo texto
+	Leitura e escrita

Em `conteudo.append(arquivo.readline().replace("\n",""))`, lemos a primeira linha do arquivo, que contém a quantidade de vitórias do usuário e removemos o "ENTER" do final. Em

`conteudo.append(arquivo.readline())`, lemos a segunda linha do arquivo, que contém a quantidade de vitórias do computador.

Como as instruções estão dentro de um `try ... except FileNotFoundError ...`, caso o arquivo não exista, o que impossibilita a leitura, geramos duas posições com o valor zero para iniciarmos um novo placar. Vale lembrar que a primeira posição, `conteudo[0]`, são as vitórias do usuário e `conteúdo[1]` são as vitórias do programa.

Caso o vencedor seja o usuário (`fim == 1`), adicionamos a nova vitória para ele com `conteudo[0] = str(int(conteudo[0]) + 1)`. Como conteudo é do tipo `string`, convertemos para inteiro para realizar a soma e, depois, reconvertemos tudo para string de novo. O mesmo acontece se a vitória for do programa (`fim == 2`).

Com o número de vitórias atualizado, resta apenas gravar tudo no arquivo. Fazemos isso com o bloco:

```
arquivo = open(str(chat_id) + ".txt","w")
arquivo.write(conteudo[0] + "\n")
arquivo.write(conteudo[1])
arquivo.close()
```

A linha `arquivo.write(conteudo[0] + "\n")` deve levar um ENTER no final (`"\n"`) para que, no processo de leitura que já vimos, possamos identificar as vitórias do usuário (que ficarão na primeira linha) e do programa (que ficarão na segunda linha).

Vamos adicionar também um comando `PLACAR`, para que o usuário possa verificar a quantidade de vitórias dele e do programa:

```
if mensagem.upper() == "PLACAR":
    conteudo = []
    arquivo = open(str(chat_id) + ".txt","r")
    conteudo.append(arquivo.readline().replace("\n",""))
    conteudo.append(arquivo.readline())
    bot.sendMessage(chat_id,"Placar:\nVocê - " + conteudo[0] +
"\nEu - " + conteudo[1])
```

Para obter esses dados, é só realizarmos a leitura do arquivo novamente e enviarmos os dados para o usuário com `bot.sendMessage`.

Vamos testar?

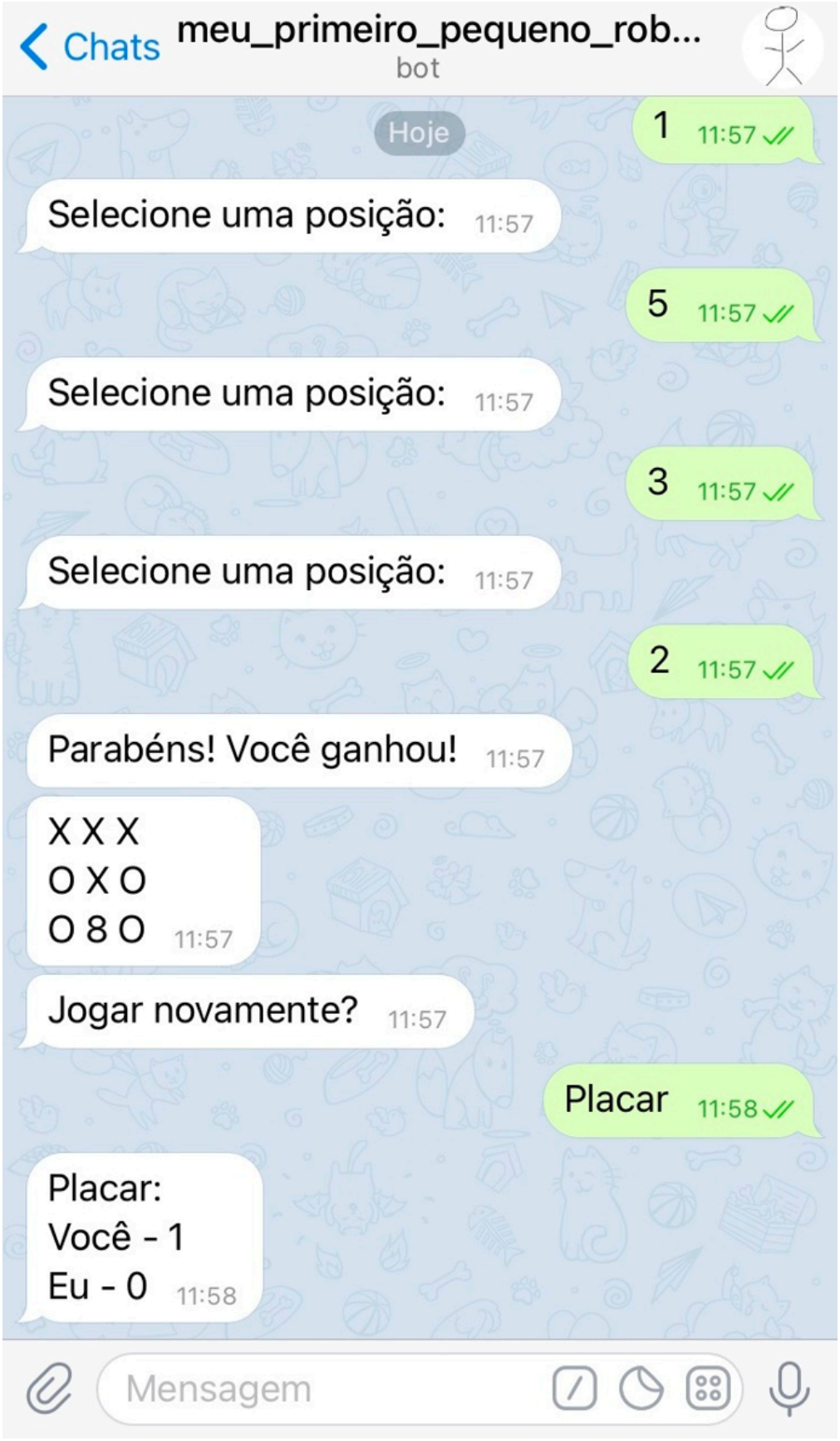

Chats
meu_primeiro_pequeno_rob...
bot
Hoje
1 11:57
Selecione uma posição: 11:57
5 11:57
Selecione uma posição: 11:57
3 11:57
Selecione uma posição: 11:57
2 11:57
Parabéns! Você ganhou! 11:57
X X X
O X O
O 8 O 11:57
Jogar novamente? 11:57
Placar 11:58
Placar:
Você - 1
Eu - 0 11:58
Mensagem

Figura 8.8: Vencendo uma partida e verificando o placar

Agora, sim, é só diversão!

E se usarmos banco de dados em um chatbot? Vamos descobrir no próximo capítulo. ;)

Capítulo 9

Biscoito da sorte – usando banco de dados

Com uma conexão a um banco de dados, praticamente tudo é possível. Você pode criar chatbots para oferecer informações sobre produtos, realizar cotações automatizadas, dar respostas automáticas para consultas de chamados técnicos, remessa de produtos, guardar dados estatísticos sobre as conversas e o uso do seu chatbot etc. Enfim, um banco de dados em um programa abre infinitas possibilidades para nós, programadores e programadoras.

A ideia aqui é criarmos um chatbot simples que, a partir do envio da palavra "sorte" em uma conversa, busque aleatoriamente uma frase no banco de dados e a envie para o usuário (ou grupo, se for o caso).

A partir deste ponto, vamos assumir que você tem algum conhecimento para instalação do gerenciador de banco de dados. Usamos o MariaDB (www.mariadb.org) para criar esse chatbot, mas você poderá usar qualquer outro que achar melhor e que esteja de acordo com sua arquitetura de software. Também consideraremos que você tem um algum conhecimento para o uso da linguagem SQL.

Ainda antes de começarmos, precisamos instalar o conector para o banco de dados, neste caso o MariaDB. A biblioteca **mariadb** nos fornece um conjunto de objetos e métodos para conexão e interação com o banco de dados.

No Windows:

```
pip3 install mariadb
```

No Linux:

```
pip install mariadb
```

Se houver dúvida sobre como fazer isso, dê uma olhada novamente no capítulo 1, *O que é necessário para continuar*. Caso esteja usando qualquer outro gerenciador de banco de dados, você deverá instalar o conector adequado para ele.

Vamos aos detalhes.

Com o gerenciador de banco de dados instalado, vamos criar um banco de dados. Para isso, usamos o seguinte script:

```
create database biscoito;
use biscoito;
create table `mensagens` (`numero` int not null
auto_increment,`texto` varchar(250), primary key(`numero`));
```

O que foi feito?

Com a linha `create database biscoito;` criamos um banco de dados com o nome de "biscoito". Já com `use biscoito;` selecionamos esse banco de dados para trabalho.

Com a linha do comando `create table`, criamos uma tabela com o nome de `mensagens` e com dois campos. O primeiro campo, `numero`, será o número da mensagem, composto por números inteiros e de autoincremento (a cada nova mensagem, o campo `numero` receberá um número automaticamente na sequência crescente). O segundo campo, `texto`, conterá um texto de, no máximo, 250 caracteres.

Para inserir mensagens, usamos as seguintes linhas:

```
insert into mensagens (texto) values ("A vida trará coisas boas
se tiveres paciência.");
insert into mensagens (texto) values ("Uma bela flor e
incompleta sem suas folhas.");
insert into mensagens (texto) values ("Espere pelo mais sábio
dos conselhos: o tempo.");
insert into mensagens (texto) values ("Todas as coisas são
difíceis antes de se tornarem fáceis.");
```

```
insert into mensagens (texto) values (“Nós somos o que pensamos.”);
```

Para consultar diretamente no banco de dados, podemos usar a linha SQL:

```
select * from mensagens;
```

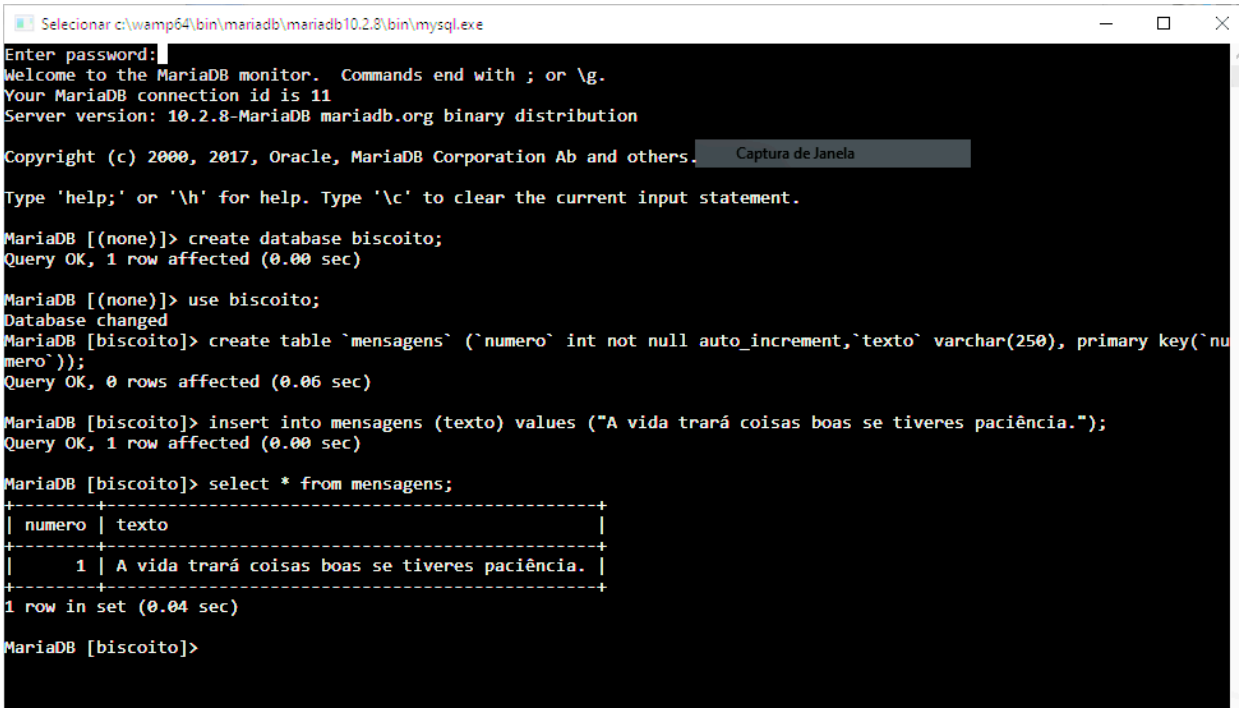


Figura 9.1: O resultado da consulta SQL diretamente no console do banco de dados

É importante que você já tenha instalado o gerenciador de banco de dados, pois ele será utilizado a partir deste ponto. Agora que está tudo pronto, vamos realizar o chatbot!

A primeira coisa de todo bot é fazer o cabeçalho e importar as bibliotecas necessárias:

```
#!/usr/bin/python3
#coding: utf-8
import time, sys
import telepot
```

```
import random
import mariadb
```

A biblioteca `random` provê uma série de comandos para geração de números aleatórios e a usaremos para sortear uma mensagem do banco de dados "biscoito". Lembra que a gente criou uma chave chamada `número` que se autoincrementa a cada cadastro de nova mensagem? Então, esse número servirá para nos ajudar a sortear uma mensagem a ser enviada para o usuário como biscoito da sorte.

Como sempre, caso esteja em uma rede em que, para se conectar à internet seja necessário o uso de usuário e senha para um proxy, lembre-se de incluir essa configuração no seu chatbot:

```
telepot.api.set_proxy('192.168.0.1:3821',('usuário','senha'))
```

Na sequência vem o bloco de programa com o qual tratamos as mensagens recebidas:

```
def handle(msg):
        content_type, chat_type, chat_id = telepot.glance(msg)

        if content_type == 'text':
                chat_id = msg['chat']['id']
                command = msg['text']
                if command.upper() == 'SORTE':
                        maria = mariadb.connect(
                                host="localhost",
                                port=3307,
                                user="root",
                                passwd="123456",
                                database="biscoito"
                                )
                        cursor = maria.cursor()
                        cursor.execute("select count(*) from
mensagens;")
                        qtde = cursor.fetchone()
                        linha = random.randint(0,qtde[0])
                        cursor.execute("select * from mensagens
where numero= " + str(linha) + ";")
                        mensagem = cursor.fetchall()
                        cursor.close()
                        maria.close()
                        bot.sendMessage(chat_id,mensagem[0][1])
```

```
                if command == '?':

bot.sendMessage(chat_id,"*Escreva:*\n_Sorte_ que eu mando um
biscoito da sorte.",parse_mode='Markdown')
```

Observe o trecho:

```
def handle(msg):
        content_type, chat_type, chat_id = telepot.glance(msg)

        if content_type == 'text':
                chat_id = msg['chat']['id']
                command = msg['text']
```

As instruções desse trecho de comandos já são velhas conhecidas para nós. Recebemos a mensagem pelo parâmetro `msg` e separamos as informações usando o `telepot.glance(msg)` em tipo de conteúdo, tipo de chat e a identificação do chat (*chat id*).

Se o tipo do conteúdo for texto, é o que estamos procurando. Nesse caso, guardamos a identificação do chat em `chat_id` e o conteúdo da mensagem em `command`.

Na sequência do código, verificamos se o conteúdo de `command`, convertido para letras maiúsculas através da instrução `command.upper()`, é igual a `SORTE`. Se sim, conectamos ao banco de dados, sorteamos uma mensagem e a enviamos para o usuário – tudo isso com:

```
if command.upper() == 'SORTE':
    maria = mariadb.connect(
        host="localhost",
        port=3307,
        user="root",
        passwd="123456",
        database="biscoito"
    )
    cursor = maria.cursor()
    cursor.execute("select count(*) from mensagens;")
    qtde = cursor.fetchone()
    print(qtde)
    linha = random.randint(1,qtde[0])
    print(linha)
```

```
    cursor.execute("select * from mensagens where numero= " +
str(linha) + ";")
    mensagem = cursor.fetchall()
    print(mensagem)
    print(mensagem[0][1])
    cursor.close()
    maria.close()
    bot.sendMessage(chat_id,mensagem[0][1])
```

Vamos aos detalhes. Primeiro, o seguinte trecho:

```
maria = mariadb.connect(
    host="localhost",
    port=3307,
    user="root",
    passwd="123456",
    database="biscoito"
)
```

Com ele, estabelecemos a conexão com o banco de dados conforme as configurações passadas. Em `host` deve ir o endereço IP de onde o banco de dados está; em `port`, a porta na qual o banco de dados vai responder – esse é um parâmetro especialmente importante se você tiver outros gerenciadores de banco de dados instalados em seu computador/servidor.

No parâmetro `user` deve ir o usuário que tem permissão de conexão ao seu banco de dados e, no parâmetro `passwd`, a senha do seu usuário.

Já no parâmetro `database` deve constar o nome do banco de dados que vamos conectar. Lembre-se de trocar esses parâmetros para que atenda à sua realidade de instalações.

Feita a conexão, precisamos primeiro criar um cursor. Fazemos isso com `cursor = maria.cursor()`. O `cursor` é que será a interface entre seu programa e o banco de dados.

Com tudo isso na mão, a primeira coisa é saber quantos registros a tabela tem – para enviar uma mensagem diferente a cada vez para o

usuário, teremos que sortear um número aleatório entre 1 e o número máximo de registros.

Para resolver esse problema, basta realizar uma consulta bem simples com `cursor.execute("select count(*) from mensagens;")`, que retornará exatamente quantos registros a nossa tabela tem.

Como o retorno será apenas um registro com a quantidade de registros, podemos apenas realizar a busca exatamente para o primeiro registro no cursor com `cursor.fetchone()`, armazenando o retorno na variável `qtde` (quantidade).

O MariaDB no Python sempre retornará, a partir de buscas (`fetch`) no cursor, tuplas com o conteúdo selecionado via SQL. No caso de `select count(*) from mensagens;`, o retorno será, por exemplo, a tupla `(100,)`, o que indica que há 100 registros na tabela.

O cursor é um *dataset* que pode ser imaginado como uma matriz na qual as linhas são os registros retornados e as colunas, os campos de cada registro.

Considerando isso, para sortearmos um número aleatório entre 1 e a quantidade de registros na tabela, usamos `linha = random.randint(1,qtde[0])`. Isso fará com que seja escolhido um número aleatório inteiro entre `1` e `qtde[0]` (primeira coluna retornada do dataset `cursor` para a variável `qtde`, que contém o valor da quantidade de registros na tabela).

Com isso em mãos, selecionamos a mensagem correspondente ao número sorteado com `cursor.execute("select * from mensagens where numero= " + str(linha) + ";")`.

Mesmo sabendo que esse comando SQL vai retornar apenas um registro, pois o campo `numero` não se repete na tabela `biscoito`, usamos `mensagem = cursor.fetchall()`, que retorna todos os registros buscados. E, após fechar o cursor e a conexão com o banco

de dados usando `cursor.close()` e `maria.close()`, enviamos a mensagem de volta para o usuário com `bot.sendMessage(chat_id,mensagem[0][1])`.

Com `mensagem[0][1]`, enviamos o primeiro registro (linha 0 da matriz) e o texto da mensagem, que está no segundo campo do registro, porque o primeiro é justamente o número da mensagem.

Lembre-se de alterar o token para seu chatbot. O programa completo ficará assim:

```
#!/usr/bin/python3
#coding: utf-8
import time, sys
import telepot
import random
import mariadb

# telepot.api.set_proxy('192.168.0.1:3128 ',('usuário','senha'))

def handle(msg):
        content_type, chat_type, chat_id = telepot.glance(msg)

        if content_type == 'text':
                chat_id = msg['chat']['id']
                command = msg['text']
                print(command)
                if command.upper() == 'SORTE':
                        maria = mariadb.connect(
                                host="localhost",
                                port=3307,
                                user="root",
                                passwd="123456",
                                database="biscoito"
                                )
                        cursor = maria.cursor()
                        cursor.execute("select count(*) from
mensagens;")
                        qtde = cursor.fetchone()
                        print(qtde)
                        linha = random.randint(1,qtde[0])
                        print(linha)
                        cursor.execute("select * from mensagens
where numero= " + str(linha) + ";")
                        mensagem = cursor.fetchall()
```

```
                        print(mensagem)
                        print(mensagem[0][1])
                        cursor.close()
                        maria.close()
                        bot.sendMessage(chat_id,mensagem[0][1])

bot =
telepot.Bot('1510435655:AAHU99c_gxEHMtNrYmBJcQq_ao4iVRFHL4o')
bot.message_loop(handle)

while 1:
        time.sleep(10)
```

Para testar, basta enviar no Telegram a palavra "sorte" para o seu bot.

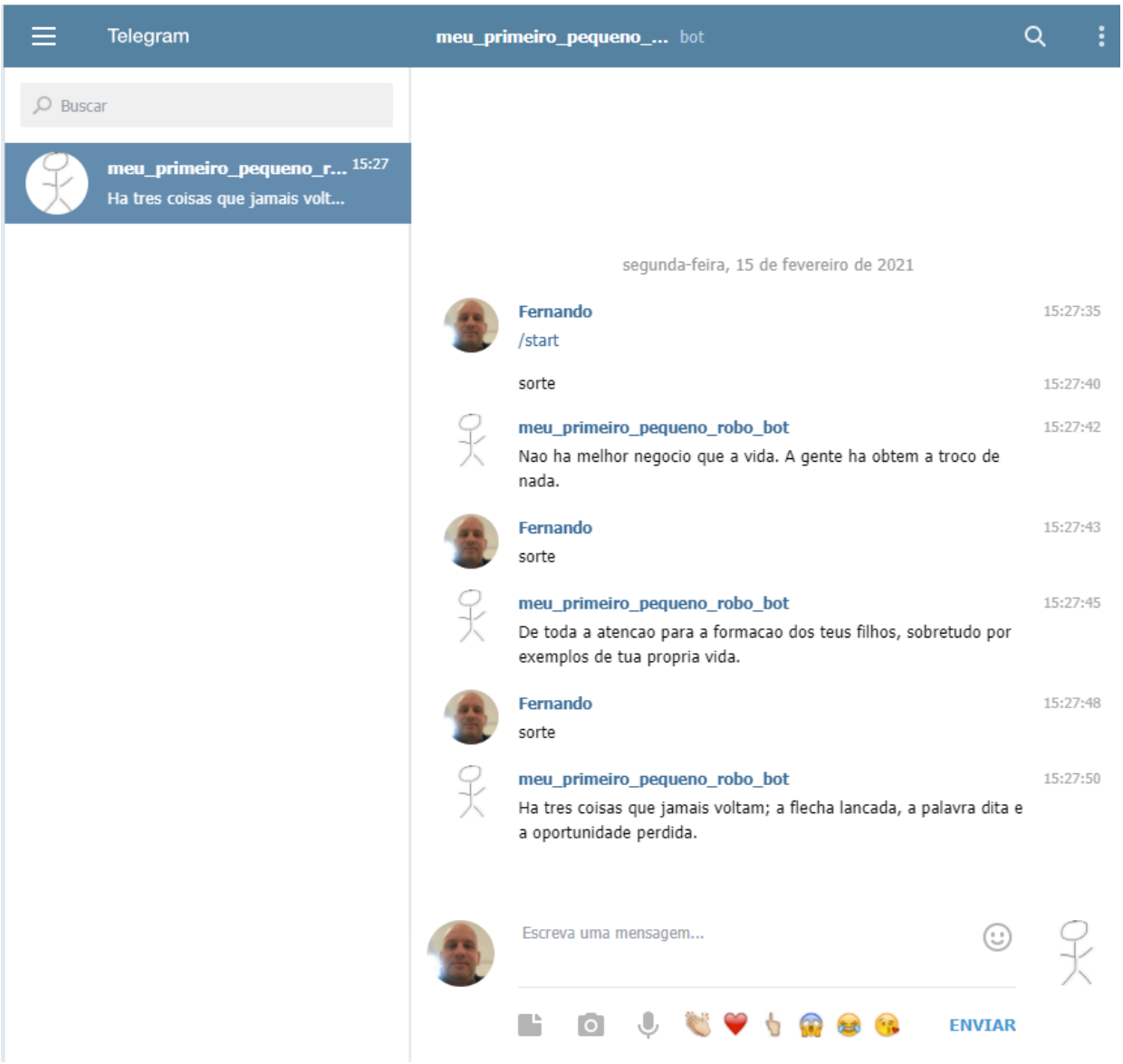


Figura 9.2: Chatbot respondendo a palavra "sorte"

Agora basta inserir mais mensagens na sua tabela e se divertir!

Capítulo 10

Conclusão

Chegamos ao fim!

Neste livro, trabalhamos a implementação de chatbots no Telegram usando a linguagem Python.

E por que chatbots no Telegram? Porque o Telegram é um dos aplicativos que mais cresce no mundo atualmente e ele se tornou uma ferramenta importante para a interação entre pessoas e entre empresas e clientes.

Os chatbots que fizemos possuem capacidade de uso de dados da internet para receber e enviar imagens, trabalhar com localização e acesso a banco de dados. Com esses princípios, é possível criar inúmeras funcionalidades como jogos, divulgação de produtos e serviços, conversores de arquivos, tradutores, rastreadores de veículos, ou qualquer outra coisa que sua imaginação conseguir inventar.

Esperamos que este livro lhe tenha sido útil e que, a partir do que foi aprendido aqui, você consiga colocar em prática as suas ideias.